Improvements for the XL Algorithm with Applications to Algebraic Cryptanalysis

Vom Fachbereich Informatik der
Technischen Universität Darmstadt genehmigte

Dissertation

zur Erlangung des Grades
Doktor rerum naturalium (Dr. rer. nat.)

von

M.Sc. Wael Said Abdelmageed Mohamed

geboren in Qaliubiya, Ägypten.

Referenten:	Prof. Dr. Johannes Buchmann
	Prof. Dr. Jintai Ding
Tag der Einreichung:	31. März 2011
Tag der mündlichen Prüfung:	06. Juni 2011
Hochschulkennziffer:	D 17

Darmstadt 2011

Wissenschaftlicher Werdegang

März 2007 - heute

Promotionsstudent am Lehrstuhl von Professor Johannes Buchmann, Fachgebiet Theoretische Informatik, Fachbereich Informatik, Technische Universität Darmstadt

Oktober 2004 - heute

Lehrbeauftragter an der Fakultät für Informatik und Informationssysteme, Fachbereich Informatik, Zagazig Universität, (Zagazig, Ägypten)

Dezember 1998 - Oktober 2004

Ausbilder an der Fakultät für Informatik und Informationssysteme, Fachbereich Informatik, Zagazig Universität (Zagazig, Ägypten)

September 2004

Master of Science in Informatik an der Fakultät für Informatik und Informtationssysteme, Fachbereich Informatik, Helwan Universität (Helwan, Ägypten)

Juli 2000 - September 2004

Masterstudium der Informatik an der Fakultät für Informatik und Informationssysteme, Fachbereich Informatik, Helwan Universität (Helwan, Ägypten)

September 1999 - Juni 2000

Besuch vorbereitender Kurse für das Masterstudium an der Fakultät für Informatik und Informationssysteme, Fachbereich Informatik, Helwan Universität (Helwan, Ägypten)

Juni 1997

Bachelor of Science in Mathematik und Informatik an der Fakultät für Naturwissenschaften, Minoufiya Universität (Shibin el Kom, Ägypten)

September 1993 - Juni 1997

Studium der Reinen Mathematik und Informatik an der Fakultät für Naturwissenschaften, Minoufiya Universität (Shibin el Kom, Ägypten)

Acknowledgements

It is very difficult to find the appropriate words that could express my heartfelt appreciation to all thoughtful people whose have shared and offered me their knowledge, help, support, experience, care and prayers. In this regard, I'll try to formulate words of thanks to several people while I would like to ask those whom I might forget to forgive me.

On the top of the list is my supervisor **Prof. Dr. Johannes Buchmann**. The opportunity to work with him has been tremendously rewarding. Throughout my four years in Darmstadt as a PhD student, Prof. Buchmann has been a source of trusted advice. I express my sincere thanks and gratitude to him for giving me the opportunity to be one of his research group, for his steady support, for his patience and for his encouragement until the realizing of this thesis. All words that I could say for you greatest Buchmann never be enough.

It is my privilege to thank my second supervisor **Prof. Dr. Jintai Ding**. He is the mastermind and the technical director of our team work in multivariate-based cryptography. All thanks and appreciation to him for the continued support of new ideas, for useful discussions and for promoting this thesis.

A special thanks go to **Dr. Stanislav Bulygin** who became my advisor in the last two years of my Ph.D. for the continues useful discussions, for the regular meetings and for the efforts to accomplish this thesis better than I could do alone. Many thanks also to my former advisor **Dr. Ralf-Philipp Weinmann**.

In addition, I'd like to thank my current and past colleagues in the research group "Cryptography and Computer Algebra, (CDC)" at the Technischen Universität Darmstadt for providing such a nice working atmosphere, for the friendly collaborations and for the useful weekly seminars. In this regard, I would like to thank **Mohamed Saied Emam** who is my colleague in the department of Computer Sciences at Zagazig University, my colleague in the CDC and my partner in the fist two years of my Ph.D. work.

I would like to express sincere gratitude to my wife, **Mayada**, for her efforts with me and for her continuous encouragement. I'm very grateful to my son, **Saifeldin**, and my daughters **Mennatullah** and **Hanin**. I apologize to all of you for lack of my presence with you most of the time.

Of course special thanks to my friend **Baher El-Sharkawy** for reading and reviewing my thesis. You are very supportive and helpful

Last, but certainly not least, I would like to thank the **Department of Computer Sciences at Zagazig University** for giving me this opportunity to do research. The generous financial support of the **Egyptian Cultural Affairs and Missions Sector** is gratefully acknowledged. The **Egyptian Cultural and Educational Office** in Berlin is acknowledged.

Wael Mohamed
Darmstadt, March 2011

Zusammenfassung

Das Lösen von Systemen multivariater Polynomgleichungen über endlichen Körpern ist ein klassisches und fundamentales Problem. Dieses Problem hat wichtige Anwendungen in verschiedenen Gebieten. In der Kryptographie beruht die Sicherheit multivariater Public-Key-Kryptosysteme auf der Schwierigkeit, Systeme multivariater quadratischer Polynomgleichungen über endlichen Körpern zu lösen.

In der Vergangenheit wurden mehrere Algorithmen zum Auffinden von Lösungen multivariater Polynomgleichungen über endlichen Körpern vorgeschlagen. Im Jahr 2000 wurde der XL Algorithmus als Werkzeug für das Lösen solcher Systeme eingeführt. Die Effizienz von XL hängt hauptsächlich von dem Grad ab, bei dem eine Lösung gefunden werden kann. In der Praxis sind Laufzeit und Speicherverbrauch des XL-Algorithmus größer als für den F4-Algorithmus, den besten bekannten Algorithmus für das Lösen von Systemen polynomialer Gleichungen. Der Hauptzweck dieser Arbeit ist es, den XL-Algorithmus zu verbessern und die vorgeschlagenen Verbesserungen an Anwendungen aus der algebraischen Kryptanalyse zu testen.

Eine Möglichkeit, den XL-Algorithmus zu verbessern, besteht darin, neue Polynome niedrigen Grades zu generieren, die im von den ursprünglichen Polynomen erzeugten Ideal liegen. Man hofft, dass diese neuen Polynome von den bestehenden linear unabhängig sind, so dass der Grad, bei dem XL das System lösen kann, minimiert werden kann. Diese Polynome kleinen Grades wurden von Jintai Ding entdeckt und als *mutants* bezeichnet. Im Prinzip ist die Verwendung dieser *mutants* die erste Verbesserung des XL Algorithmus, die in dieser Arbeit vorgeschlagen wird. Dies wird im MutantXL Algorithmus und seiner Implementierung erreicht. Eine weitere Verbesserung des MutantXL-Algorithmus namens MXL2 wird ebenfalls in dieser Arbeit beschrieben. MXL2 verwendet zwei wesentliche Verbesserungen über $\mathbb{F}_2$, die das Lösen von Systemen mit deutlich kleineren Matrizen erlauben als XL und MutantXL. MXL2 wird in dieser Arbeit im Rahmen der algebraischen Kryptanalyse benutzt, um zwei multivariate public-key Kryptosysteme zu brechen, nämlich Little Dragon Two und Poly Dragon.

Die zweite Verbesserung hängt mit der Struktur der von XL generierten Matrizen zusammen. Ab einem gewissen Grad tendieren diese Matrizen dazu, schwach besetzt zu sein. Deshalb ist es bezüglich Speicher und Zeit sehr kostspielig, diese Matrizen mit Gauss-Elimination auf Zeilenstufenform zu bringen. Die Verwendung des Wiedemann-Algorithmus anstatt der Gauss Elimination über $\mathbb{F}_{256}$ mit einer skalaren Version des Wiedemann-Algorithmus wurde von Bo-Ying Yang et al. eingeführt. In dieser Arbeit beschreiben wir den Gebrauch des blockweisen Wiedemann-Algorithmus über $\mathbb{F}_2$ und seine Kombination mit dem XL-Algorithmus, die wir als WXL bezeichnen. Eine Möglichkeit, den WXL Algorithmus zu verbessern, besteht darin, mehr als einen Prozessor zu verwenden. Man nutzt dabei die Tatsache aus, dass der Wiedemann-Algorithmus parallelisiert werden kann. Indem man PWXL, eine parallelisierte Version von WXL, benutzt, können Systeme mit einer größeren Zahl von Variablen gelöst werden. Diese Systeme wurden bisher von keinem anderen algebraischen Algorithmus gelöst. PWXL kann insbesondere Instanzen des HFE Kryptosystems mit 37 quadratischen Gleichungen in 37 Variablen über $\mathbb{F}_2$ lösen. Diese besitzen

den gleichen univariaten Grad wie die von HFE Challenge-2 und können bei Benutzung von 81 Prozessoren in 7.5 Tagen erfolgreich gelöst werden.

Die Kombination der beiden vorgeschlagenen Verbesserungen, der *mutant* Strategie und des parallelisierten Wiedemann-Algorithmus, ist die dritte Verbesserung. Der erste Teil dieser Verbesserung ist das Erzeugen der *mutants* durch Benutzung des Kerns einer Matrix, der ebenfalls mit Hilfe des Wiedemann-Algorithmus erzeugt werden kann. Der zweite Teil besteht darin, die durch den Wiedemann-Algorithmus erzeugten *mutants* für das Lösen des Systems zu verwenden. Das Erzeugen von *mutants* durch Wiedemann und das Lösen mit MutantXL ist das erste Szenario für den Gebrauch solcher *mutants*. Das zweite Szenario ist die Verwendung des WMXL Algorithmus, der eine Kombination aus XL-Algorithmus, Wiedemann-Algorithmus und dem Konzept der *mutants* darstellt, um für strukturierte Systeme eine Lösung auf effektive Weise zu erhalten.

Es werden effiziente und wirksame Verbesserungen des XL-Algorithmus vorgestellt. Diese Verbesserungen basieren einerseits auf dem Konzept der *mutants* und andererseits auf dem parallelisierten Wiedemann-Algorithmus. Die Bedeutung dieser Verbesserungen wird anhand der Lösung von Systemen multivariater Polynomgleichungen, die ihren Ursprung in der Kryptographie haben, aufgezeigt. Deshalb sollten Designer neuer Kryptosysteme diesen neu vorgeschlagenen Algorithmen Beachtung schenken.

Abstract

The problem of solving systems of multivariate polynomial equations over finite fields is a classical and fundamental problem in symbolic computation. This problem has important applications in numerous domains. In cryptography, the security of multivariate-based public-key cryptosystems relies on the difficulty of solving systems of multivariate quadratic polynomial equations over finite fields.

Several algorithms have been proposed to find solution(s) for systems of multivariate polynomial equations over finite fields. In 2000, the XL algorithm was introduced as a tool for solving such systems. The overall performance of XL depends on the degree at which a solution could be found. From a practical point of view, the running time and memory consumption for XL is greater than consumptions of the F4 algorithm, the best known efficient algorithm for solving systems of polynomial equations. The main purpose of this thesis is to improve the XL algorithm and test the suggested improvements with applications to algebraic cryptanalysis.

One way to improve the XL algorithm is to generate new low degree polynomials that are in the ideal generated by the original polynomials. The hope is that these new polynomials are linearly independent from the ones already generated by XL, so that the degree at which XL has the ability to solve a system could be minimized. These low degree polynomials are discovered and named *mutants* by Jintai Ding.

Basically, the use of these mutants is the first improvement that is suggested in the thesis at hand. This improvement is the MutantXL algorithm and its implementation. An improvement of MutantXL, called MXL2, is also presented in this thesis. MXL2 uses two substantial improvements over $\mathbb{F}_2$ that oftentimes allow to solve systems with significantly smaller matrix sizes than XL and MutantXL. The use of MXL2 in the context of algebraic cryptanalysis is demonstrated by breaking two multivariate-based public-key cryptosystems, namely Little Dragon Two and Poly-Dragon, more efficiently than F4.

The second improvement is related to the structure of matrices generated by XL. These matrices tend to be sparse. Therefore, transforming such matrices to the row echelon form using Gaussian elimination makes the linear algebra cost so much in terms of memory and time due to the fill-in property. The use of the Wiedemann algorithm as a solver instead of Gaussian elimination over $\mathbb{F}_{256}$ with a scalar version of the Wiedemann algorithm was introduced by Bo-Yin Yang et al. In this thesis, we represent the use and combination of the block Wiedemann algorithm over $\mathbb{F}_2$ and XL, referred to as WXL algorithm. One way to improve WXL is to use more than one processor based on the advantage that the Wiedemann algorithm is able to be parallelized. By using PWXL, a parallelized version of WXL, systems with higher number of variables can be solved. These systems were not solved before by any other algebraic solver. In particular, PWXL can successfully solve instances of the HFE cryptosystem over $\mathbb{F}_2$ that have the same univariate degree as the HFE challenge-2 with 37 quadratic equations in 37 variables using 81 processors in 7.5 days.

The combination of the two suggested improvements for the XL algorithm, the mutant strategy

and the parallelized Wiedemann algorithm, is the third improvement. The first part of this improvement is the generation of mutants by using the matrix kernel which also could be computed by the Wiedemann algorithm. The second part is to use mutants that are generated by Wiedemann algorithm. Generating mutants using the Wiedemann algorithm then solving using MutantXL is the first scenario to use such mutants. The second scenario is the use of the WMXL algorithm which is a combination of the PWXL algorithm and the concept of a mutant in order to obtain a solution in an efficient way for structured systems.

Effective and efficient improvements for the XL algorithm are presented. These improvements are based on the concept of a mutant and the parallelized Wiedemann algorithm. The importance of these improvements is indicated by solving systems of multivariate polynomial equations arising from cryptography. Therefore, designers of new cryptosystems should take care about these new suggested algorithms.

Contents

1 Introduction and Motivation

This chapter outlines the work that is presented in this thesis. We present the motivation, problem statement, research statement, research questions, research objectives and contributions of our work.

1.1 Preface

Today, we are living in the era of information revolution that has left its clear reflections and implications to all the sectors of our society. Indeed, we are moving towards an information society in which information is playing an increasingly important role in our lives. The rapid development of information technology and telecommunication leads to different forms and large amount of information. Based on the value of the information, the protection of this information either in transit or in storage is considered a very important process. The appropriate security measures that are taken to ensure the protection of information from all kinds of manipulation whether accidental or intentional is the fundamental concept of the term information security. In order to transform this concept into reality, we need cryptography which is the most fundamental information security tool.

Simply speaking, cryptography is the art of protecting information by transforming it into an unreadable form using some secret. Only those who possess the secret can transform information back to its original form. Cryptography can be classified into secret-key cryptography and public-key cryptography.

The main feature of public-key cryptography is the use of two keys, a public key and a private key. It is not possible to figure out what the private key is, given only the public key. The public key is used to encrypt a message while only the corresponding private key is used to decrypt that message. Public-key cryptography is necessary for using Internet in a secure manner. The security of current public-key cryptosystems (RSA, ElGamal) is based on the computational difficulty of solving certain hard problems. These problems will not be hard any more under the assumption that quantum computers with enough quantum bits exist. Therefore, current traditional cryptosystems are not secure under this assumption.

We need new alternative cryptosystems that have the potential to resist the quantum attacks. One of such alternatives is the multivariate-based cryptography which refers to public-key cryptosystems that are based on the problem of solving multivariate polynomial equations over finite fields. This problem is NP-complete and it is believed that quantum computing does not give a significant advantage in solving it. Multivariate-based cryptography can be used for encryption or signatures as well as authentication in an asymmetric way. Furthermore, it has the potential to be used in applications for small devices with limited computing power. Despite all these advantages, almost all the proposed multivariate encryption schemes are broken while few multivariate signature schemes are still secure. In order to gain the advantages of multivariate-based cryptography,

more effort in this research field is required to design efficient and secure multivariate-based cryptosystems. In this context, we can use cryptanalysis in order to find some weakness or insecurity in a cryptosystem.

In cryptography, constructing a system of multivariate polynomial equations that describes a cryptographic primitive in which the secret is represented by unknown variables and then solving this system to recover that secret information is called algebraic cryptanalysis. The most famous two algorithms that are used for solving systems of multivariate polynomial equations are the F4 algorithm for computing Gröbner bases and the XL algorithm.

The importance and significance of the F4 algorithm comes from its ability to break the HFE challenge-1 in 2002 by Faugère and its better performance compared to the XL algorithm in most cases. The significance of XL in the cryptographic community comes from it simplicity. It is known that XL can be represented as a variant of F4 for solving systems with unique solution. The question that arises in mind is as follows: Can the simpler approach of XL be made more efficient? In this context, improving the XL algorithm is a promising research area. In fact, we need effective and efficient improvements for the XL algorithm with applications to algebraic cryptanalysis.

1.2 Motivation

In cryptanalysis, the basic principle of algebraic cryptanalysis goes back to Shannon 1949. He stated that breaking a good cipher should require "as much work as solving systems of simultaneous equations in a large number of unknowns". Furthermore, the AES block cipher as well as most block and stream ciphers can be described by an extremely overdetermined multivariate quadratic system over finite fields. Therefore, algebraic cryptanalysis is considered as a generic attack that is applicable to multivariate-based public key cryptosystems, block ciphers, stream ciphers and hash functions.

On the other hand, sparse linear algebra was successfully used in integer factorization specially the use of block Wiedemann algorithm over $\mathbb{F}_2$ in the number field sieve method to factor the RSA-768 number. It is known that, the Wiedemann algorithm is one of the three well-known methods that are adapted well to sparse matrices. The other two algorithms are Conjugate Gradient (CG) and Lanczos. The great potential of using the Wiedemann algorithm in integer factorization leads to the importance of using this algorithm as a solver instead of Gaussian elimination. Moreover, the Wiedemann algorithm can be efficiently parallelized.

We are motivated to construct efficient tools for solving systems of quadratic multivariate polynomial equations over $\mathbb{F}_2$ that aim to produce methods for assessing the security of multivariate-based public key cryptosystems. We believe that algebraic cryptanalysis will become one of the most important attacks in the next few years, so we try to investigate the design criteria which may guarantee a high resistance to algebraic cryptanalysis.

1.3 Problem Statement

Public-key cryptography is a fundamental tool that enables secure transmission of information on the Internet. The security of current public-key cryptosystems is based on two hard computational problems, namely, factoring large integers and computing discrete logarithm. These two

problems were proven to be solved in polynomial time using a quantum computer. Therefore, we need to construct strong alternatives to current traditional cryptosystems that are able to resist the quantum computing based attacks. One of these promising alternatives is the multivariate-based cryptography. In order to construct strong multivariate cryptographic schemes, we should be able to understand the reasons behind the weakness of such schemes. One way to find some weakness of a multivariate-based public key cryptosystem is the use of algebraic cryptanalysis which in turn depends on the problem of solving systems of multivariate polynomial equations over finite fields.

Algebraic cryptanalysis is not only applicable to multivariate-based cryptography, but also it is applicable to symmetric ciphers. By representing a symmetric cipher as a system of simultaneous equations in a large number of unknowns then solving the resulting multivariate system is equivalent to recovering the secret unknowns. Therefore, solving systems of multivariate polynomial equations over finite fields is an important task in cryptography, mainly in cryptanalysis.

1.4 Research Statement

Currently there are many different algorithms which are based on different strategies to solve systems of multivariate polynomial equations. Gröbner basis methods and linearization are the two main categories to solve such systems of equations. The general strategy of the XL algorithm can be viewed as a combination of bounded degree Gröbner basis and linearization. The basic idea behind XL is to produce from each original polynomial a large number of higher degree polynomials by multiplying the original polynomial with all possible monomials up to some bounded degree, then XL linearizes the extended system and solves it using Gaussian elimination.

One way to improve XL is to generate new low degree polynomials that are in the ideal generated by the original polynomials with the hope that these new polynomials are linearly independent from the polynomials that are already generated by XL and the degree at which XL has the ability to solve a system could be minimized. The second way is to use the parallelized Wiedemann algorithm as a solver instead of Gaussian elimination. This is due to the fact that when XL extends the system to a certain high degree the resulting system tends to be sparse. The combination of the two suggested improvements for the XL algorithm, mutants (low degree polynomial equations) and the Wiedemann algorithm is the third suggested improvement.

In order to measure the effectiveness and efficiency of the suggested improvements, we apply them to real world applications including instances of dense random systems, instances of the HFE cryptosystem, instances of the Little Dragon Two cryptosystem and instances of the Poly-Dragon cryptosystem. The main benefit of this work is to indicate the importance of algebraic cryptanalysis. Therefore, designers of new cryptosystems should take care about algebraic cryptanalysis.

1.5 Research Questions

In the area of algebraic cryptanalysis, the intention is to answer the following questions:

- How efficiently and effectively can we solve multivariate quadratic polynomial equation systems?
- How to improve the XL algorithm?

- How can sparse linear algebra help?
- How to generate low degree polynomials using sparse linear algebra?
- How efficient is the parallelization of the Wiedemann algorithm?
- How effective is the improvement(s) for the XL algorithm?

1.6 Research Objectives

The overall objective of this thesis is to improve the XL algorithm using different strategies. In this context, other objectives are to:

- Present a design for the improved algorithms.
- Implement these algorithms using the C/C++ language.
- Test the performance of the proposed tools and compare to other solvers.
- Use these tools to attack some real world cryptosystems.
- Analyze which tool is better in what contexts.

1.7 Contributions of our Work

The work reported in this thesis is aimed at proposing tools for solving systems of quadratic multivariate polynomial equations with applications to algebraic cryptanalysis. The major contributions of our work can be listed as follows:

- Sharing the proposal of the MutantXL algorithm.
  MutantXL is an improvement for the XL algorithm that uses lower degree polynomials called mutants. These mutants help in minimizing the degree at which a solution could be found. By using mutants, MutantXL algorithm solves with significantly smaller matrix size than the XL algorithm. The discussions, design, implementation and testing for the MutantXL algorithm is a joint work with Mohamed Saied Emam Mohamed.
- Sharing the proposal of the MXL2 algorithm.
  MXL2 is an improvement for the MutantXL algorithm that uses two substantial improvements. These two improvements are based on multiplying only a necessary number of mutants instead of all mutants and extending systems of polynomials only partially to higher degrees. The MXL2 algorithm outperforms the F4 algorithm in terms of memory while for time performance it is not always the case. This work was done in collaboration with Mohamed Saied Emam Mohamed.
- Presenting the WXL algorithm.
  WXL, Wiedemann XL, algorithm is an improvement for XL that uses the block Wiedemann solver instead of Gaussian elimination. The design and implementation of WXL is presented. WXL is better than F4 in terms of memory for instances of the HFE cryptosystems and dense random systems.

- Proposing the PWXL algorithm.
  PWXL, Parallelized Wiedemann XL, algorithm is a parallelized version of the WXL algorithm. By using more than one processor and sparse matrix representation, PWXL can solve instances of the HFE cryptosystems that were never solved before by any known algebraic solver.

- Proposing the WMXL algorithm.
  WMXL, Wiedemann Mutant XL, algorithm is is an improvement for the XL algorithm that is based on the combination of the mutant strategy, the parallelized Wiedemann algorithm and the XL algorithm. WMXL and PWXL are similar in using the block Wiedemann algorithm as a solver. WMXL and MXL2 as well as MutantXL are similar in using the concept of mutants. The generation of mutants using the block Wiedemann algorithm is also contributed. By using WMXL, we are able to solve random instances of multivaraite quadratic systems more efficiently than the F4 algorithm.

- Proposing the KXL algorithm.
  KXL, Kernel-based XL, algorithm is a variant of WMXL that is used to generate linear mutants instead of solving in the last step of WMXL. Afterwards, the quadratic and linear polynomials are solved using MutantXL or MXL2. By using KXL, we are able to solve up to instances of HFE cryptosystem with 50 equations in 50 variables while F4 is able to solve up to 39 equations in 39 variables.

- Breaking the Little Dragon Two multivariate-based public-key cryptosystem.
  As an application of using MXL2, we are able to break the Little Dragon Two cryptosystem more efficiently than F4. Practically, MXL2 was able to solve an instance of Little Dragon Two with 229 equations and variables using almost 25GB RAM in 3 hours while F4 solves the same instance using 59GB RAM in 1.6 days.

- Breaking the Poly-Dragon multivariate-based public-key cryptosystem.
  We used MXL2 to perform algebraic cryptanalysis to the Poly-Dragon cryptosystem. In this regard, MXL2 solves instances of Poly-Dragon cryptosystem up to 299 equations in 299 variables while F4 can solve instances up to 259 equations in 259 variables.

The work that is presented in this thesis is a joint work with other authors. It was published in the following papers:

- Jintai Ding, Johannes Buchmann, Mohamed Saied Emam Mohamed, Wael Said Abd Elmageed Mohamed and Ralf-Philipp Weinmann, "MutantXL", In *"Proceedings of the 1st international conference on Symbolic Computation and Cryptography (SCC08)", Beijing, China*, pp. 16–22, LMIB (2008).

- Mohamed Saied Emam Mohamed, Wael Said Abd Elmageed Mohamed, Jintai Ding and Johannes Buchmann, "MXL2: Solving Polynomial Equations over GF(2) Using an Improved Mutant Strategy", In *"PQCrypto '08: Proceedings of the 2nd International Workshop on Post-Quantum Cryptography", Cincinnati, OH, USA*, pp. 203–215, Springer, Heidelberg (2008).

- Johannes Buchmann, Jintai Ding, Mohamed Saied Emam Mohamed and Wael Said Abd Elmageed Mohamed, "MutantXL: Solving Multivariate Polynomial Equations for Cryptanalysis", In *"Dagstuhl Seminar Proceedings"*, Schloss Dagstuhl - Leibniz Center for Informatics, Germany (2009).

- Wael Said Abd Elmageed Mohamed, Jintai Ding, Thorsten Kleinjung, Stanislav Bulygin and Johannes Buchmann, "PWXL: A Parallel Wiedemann-XL Algorithm for Solving Polynomial Equations over GF(2)", In *"Proceedings of the 2nd international conference on Symbolic Computation and Cryptography (SCC 2010)" , Royal Holloway, University of London, Egham, UK*, pp. 89–100, 2010.

- Johannes Buchmann, Stanislav Bulygin, Jintai Ding, Wael Said Abd Elmageed Mohamed and Fabian Werner , "Practical Algebraic Cryptanalysis for Dragon-Based Cryptosystems", In *"Proceedings of the 9th international conference on Cryptology and Network Security (CANS 2010)" , Kuala Lumpur, Malaysia*, pp. 140–155, Springer (2010).

1.8 Thesis Structure

This thesis consists of seven chapters and one appendix. Chapter 1, the chapter in our hands, is an outline of the thesis. Chapter 7 concludes the thesis and presents a discussion to possible future work. For the remainder, the reader who is familiar with one chapter can omit it without affecting the whole meaning. At the same time, all the chapters together draw a complete picture to our work.

The remainder of this thesis is organized as follows. Chapter 2 presents the different applications to algebraic cryptanalysis that are used in this thesis as case studies, namely the HFE cryptosystem, the Little Dragon Two cryptosystem and the Poly-Dragon cryptosystem. A short overview of multivariate-based cryptography, MQ-problem and cryptanalysis of multivariate-based cryptosystems are presented. In Chapter 3, we survey the XL algorithm from creation time till nowadays. A practical limit for XL under our available resources, 128GB RAM, is demonstrated. The first suggested improvement that is based on the Ding's concept of a mutant and the breaking of Dragon-based cryptosystems are discussed and presented in Chapter 4. The second improvement that is based on the parallelized Wiedemann algorithm is the contribution presented in Chapter 5. In Chapter 6, a combination of the concept of a mutant and the parallelized Wiedemann algorithm is discussed.

one appendix is included. Appendix A presents the basic definitions that are used in this thesis.

2 Multivariate Cryptography and Algebraic Cryptanalysis

In this chapter we present different applications to algebraic cryptanalysis that are used in this thesis. We discuss the MQ-problem, multivariate-based cryptography and different methods to cryptanalyze multivariate-based cryptography. An overview of the HFE, the Little Dragon Two and the Poly-Dragon cryptosystems is introduced.

2.1 Introduction

Digitally sign documents (e-sign), electronic money transfer (e-money), electronic mail sending/receiving (e-mail), digital copyright protection and electronic voting (e-voting) are examples of computer-based systems that are the requirements of modern era. The success and efficiency of these systems depend mainly on information security which in turn depends on cryptography to protect information either in transit or in storage.

From a historical point of view, the oldest known documented use of cryptography in writing dates back to circa 1900 B.C. when an Egyptian scribe, named Khnumhotep II, used non-standard hieroglyphs in an inscription. Particularly, the scribe used hieroglyph substitutions on the wall of his tomb, even if the aim of this substitution was to impart dignity and authority, these scripts were recorded as the oldest cryptographic algorithm. Beyond this, it was indicated in a lot of publications that cryptography appeared spontaneously sometime after writing was invented, with applications ranging from diplomatic missives to war-time battle plans. Therefore, modern cryptography is the key technology that is necessary to keep up the pervasive development of computer communications.

Modern cryptography is the study of mathematical techniques related to some specific information security goals. These goals include: Confidentiality, Integrity, Non-repudiation, and Authentication. The Confidentiality, sometimes referred to as Privacy, is the process in which the information cannot be understood by anyone for whom it was unintended. The Integrity can be defined as the process in which the information cannot be modified in storage or transit between sender and intended receiver without the modification being detected. While Non-repudiation means, the creator/sender of the information cannot deny at a later stage his or her intentions in the creation or transmission of the information. The process in which the sender and receiver can confirm each other's identity and the origin/destination of the information is the definition of the Authentication goal.

In cryptography, the procedures and protocols that meet some or all of the four security goals are known as cryptographic algorithms (or cryptographic schemes). Encryption is the process of converting source information (referred to as plaintext) using a cryptographic algorithm to make it meaningless to anyone except those possessing special knowledge, usually referred to as a key.

The result of the encryption process is an encrypted information (referred to as ciphertext). Decryption is the reverse process of encryption, i.e. convert the encrypted information to be readable again using only a key. The Encryption and Decryption processes provide an achievement to the confidentiality goal. Based on the number of keys that are used for encryption and decryption, we can divide the cryptographic algorithms into two categories. The first category which is based on a single key for both encryption and decryption is called Secret-Key Cryptography (SKC), sometimes it is referred to as shared-key cryptography, conventional algorithms, or symmetric cryptography. The second category is Public-Key Cryptography (PKC) which uses one key for encryption and another for decryption, it is also referred to as asymmetric cryptography or asymmetric algorithms. The term cryptosystem is used in PKC to refer to a cryptographic algorithm that consists of three algorithms, one for key generation, one for encryption, and one for decryption. While the term cipher (sometimes cypher) is a cryptographic algorithm used in SKC that consists of only two algorithms, one for encryption and the other for decryption. The term digital signature refers to a cryptographic scheme that is similar to the handwritten signature. A digital signature scheme typically consists of three algorithms. The first one is a key generation algorithm that outputs a private key and a corresponding public key. The second one is a signing algorithm that produces a signature using a message and a private key as inputs. The third one is a signature verifying algorithm that either accepts or rejects the message's claim to authenticity. The inputs for the signature verifying algorithm are a message, public key and a signature. A successful verification means that the verifier has no doubt that the singer's private key encrypted the message.

PKC is necessary for using the Internet in a secure manner, e.g., e-Commerce applications, private communication and secure communication. The security of traditional public-key cryptosystems (RSA, ElGamal) is based on the computational difficulty of solving a certain type of hard problems. Factoring large integers is the case for RSA, while the difficulty of solving the discrete logarithm problem is the base for ElGamal and Elliptic Curve based cryptosystems.

Under the assumption that quantum computers with enough quantum bits exist and taking into account Shor's quantum algorithm for computing discrete logarithm and factorization efficiently [155, 156], traditional cryptosystems (RSA, ElGamal) are not secure anymore. Indeed, we need alternatives post-quantum cryptosystems that have the potential to resist the quantum computing based attacks. In post-quantum cryptography there exist five major alternatives to traditional cryptography that are believed to be resistant to the quantum computing based attacks. These alternatives are Hash-based cryptography, Lattice-based cryptography, Code-based cryptography, Group-based cryptography and Multivariate-based cryptography. These alternatives are believed to be resistant to Grover's quantum search algorithm [90] which could be used to solve NP-complete problems by performing exhaustive searches over the set of possible solutions.

In this thesis, we are interested in solving systems of multivariate quadratic polynomial equations that are arising from multivariate-based cryptography. The problem of solving such systems was proven to be an NP-complete problem and it is not known whether there is a polynomial algorithm for solving it on a quantum computer. The aim is to test our suggested improvements for the XL algorithm with a real world examples.

This chapter is organized as follows: Section 2.2 presents the MQ-problem. In Section 2.3, we introduce the multivariate-based cryptography. We present HFE, Little Dragon Two and Poly-Dragon which are examples of multivariate-based cryptography applications that we use. An overview of multivariate-based cryptanalysis which is the complement partner of cryptography

exists in Section 2.4. In Section 2.5, we conclude this chapter.

2.2 The MQ Problem

In this section, we consider the problem of solving a system of m quadratic multivariate polynomial equations with n variables over a finite field $\mathbb{F} = \mathbb{F}_q$ with q elements. Let $p \in \mathbb{F}[x_1, \ldots, x_n]$ be a quadratic polynomial,

$$p(x_1, \ldots, x_n) = \sum a_{i,j,k} x_j x_k + \sum b_{i,j} x_j + c_i$$

for $0 \leq j \leq k \leq n$ and $a_{i,j,k}, b_{i,j}, c_i \in \mathbb{F}$ (quadratic, linear and constant coefficients, respectively).

Let P denotes a set of m multivariate quadratic polynomial equations such that

$$P = \begin{cases} p_1(x_1, \ldots, x_n) = 0, \\ p_2(x_1, \ldots, x_n) = 0, \\ \vdots \qquad\qquad \vdots \\ p_m(x_1, \ldots, x_n) = 0 \end{cases}$$

The problem to find a common zero $z \in \mathbb{F}^n$ of the polynomial equationss $p_i(x_1, \ldots, x_n) = 0$, $1 \leq i \leq m$ is called the **MQ-problem**. When the polynomials in P are not quadratic, the problem is known as **PoSSo-problem** (**Po**lynomial **S**ystem **So**lving). We can reformulate the MQ-problem over $\mathbb{F}_2$ as a question as follows: Is it possible to find $(z_1, \ldots, z_n) \in \{0, 1\}^n$ such that $p_i(z_1, \ldots, z_n) = 0$, $1 \leq i \leq m$ where the arithmetic are as defined in $\mathbb{F}_2$?

Frankel and Yesha [79] proved that the MQ-problem over $\mathbb{F}_2$ is NP-complete problem. This proof was done by reducing the problem of solving a randomly selected instance of the 3-SAT problem [86] into the problem of solving a system of randomly selected multivariate quadratic polynomial equations over $\mathbb{F}_2$ in a polynomial time. It is known that the 3SAT problem is NP-complete. Therefore, the MQ-problem is also NP-complete. Garey and Johnson [86] referenced the proof of Frankel and Yesha as well as stated that the proof was done independently by L.G. Valiant in a private communication.

Reducing the MQ-problem over any field into an instance of the MQ-problem over $\mathbb{F}_2$ in a polynomial time is the straightforward direction to prove the NP-completeness over any field. Typically, this was the proof by Patarin and Goubin that was presented in the appendix of the extended version of [135].

It is well known that the NP-completeness of the problem is not sufficient for its use in cryptography, the problem has to be hard in average [14], randomly generated instances of the MQ-problem are hard on average.

2.3 Multivariate-based Cryptography

The multivariate-based public-key cryptosystems (MPKCs) are public-key cryptosystems (PKC) that are based on the problem of solving systems of multivariate nonlinear polynomial equations over finite fields. In order to avoid increasing the key size for MPKCs, most of the used and

suggested MPKCs are based on the problem of solving systems of multivariate quadratic polynomial equations over finite fields. This problem is called "MQ-problem" and it is NP-complete, see Section 2.2. Therefore, MPKCs are sometimes referred to as MQ-schemes. Several MPKCs that are based on the MQ-problem have been proposed in the last two decades. There are two more problems which are used in the context of constructing MPKCs. The first problem is the MinRank-problem [25] which was reformulated in the cryptographic context in [37]. The second problem is the IP-problem [134] or its extension EIP-problem [59]. In this thesis, we are interested in the MQ-problem which depends on the number of variables and the number of equations. An overview of MPKCs can be found in [57, 65].

The general idea of a typical MPKC is to construct the public key $P = T \circ P' \circ S$ which is a set of multivariate quadratic polynomials over a finite field from the private key $\{P', T, S\}$, where $\circ$ denotes the composition of functions. The private key $\{P', T, S\}$ is a set of maps P', T and S. P' is called the central map (private polynomial) which is easy to be invertible using a trapdoor. T and S represent two invertible affine transformations over finite field. In order to encrypt a plaintext x, we take this plaintext and evaluate it through the public key P. The result is $y = P(x)$ where y is the associated ciphertext. The general procedure to decrypt a message is to take a ciphertext y then convert it to the plaintext x by using the secret key $\{P', T, S\}$. This can be accomplished by applying the three inverses T^{-1}, P'^{-1} and S^{-1} to the ciphertext, the plaintext $x = S^{-1} \circ P'^{-1} \circ T^{-1}(y)$. In order to generate a signature x, we need to use the private key to find at least one pre-image of the public key. Signature verification is done by evaluating a given signature x through the public key. If the result of the verification is the same as the given message y then we accept the signature.

The variations of MPKCs depend on the variations of the structure and the number of the central maps that could be used. All the known central maps of MPKCs consist of only one map except the TTM [120] which uses two maps and the TRMS [171] scheme which uses more than two maps.

Based on the structure of the central map, MPKCs could be grouped into different classification schemes. According to [57], all existing MPKCs can be divided into two categories with one exception, the Isomorphism of Polynomial (IP) authentication scheme. The first category is called bipolar and the second is called mixed. A second classification scheme of MPKCs is defined in terms of the number of finite fields that are used to construct the public and private keys [180]. In this classification, the MPKCs are divided into two categories. The first one is called MixedField, sometimes referred to as TwoField, cryptosystems in which the public key is defined over a smaller base field and the central map which is a univariate polynomial over a large extension of the base field. MixedField cryptosystems can be used for both encryption and signature. The second category is called SingleField in which both the public key and the central map are defined over the same small base field. This category, SingleField, is usually used only for signature. Lih-Chung Wang et al. [172] presented another class of MPKCs called MediumField cryptosystems in which the public key is defined over a small base field while the central map is defined over several medium-sized field extensions. Therefore, Lih-Chung Wang et al. divided the MixedField category into another two categories known as BigField and MediumField. In some publications, for example [64], MixedField and BigField are used interchangeably. This classification can be shown in Figure 2.1.

The UOV [105] and Rainbow [61] signature schemes are examples of the SingleField category. In this thesis we are interested in MixedField category. As an example of the bipolar as well as

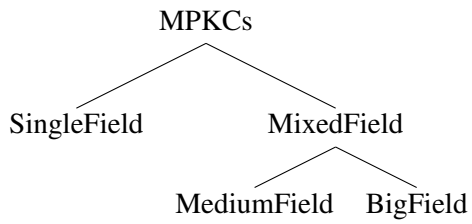

Figure 2.1: Multivariate-based Public-Key Cryptosystems Classification

MixedField category, Matsumoto and Imai [117] presented the so called C^* (or MI) cryptosystem. In cryptography, this cryptosystem is considered as the first candle in the road of MPKCs. The basic idea is to use two different finite fields, namely a base field and an extension field of some degree over the base field to hide a monomial field equation. In [132], Patarin was able to break C^*. Furthermore, Patarin suggested some cryptosystems to repair C^*, namely HFE, IP in [134] and Dragon, MIIP-3 (Matsumoto-Imai with Improved Parameters of degree 3) in [133]. There is also another suggested cryptosystems to repair C^* presented by Patarin et al. in [135, 136].

The idea of the HFE, stands for Hidden Field Equations, cryptosystem is based on hiding a polynomial instead of hiding a monomial like C^*. The main problem of this cryptosystem is that the computation of the secret key is not efficient as the original C^*. A more detailed discussion for HFE exists in Section 2.3.1. Repairing C^* with the same kind of easy secret key computations was the basic idea to introduce the Dragon as well as the MIIP-3 cryptosystems. The Dragon cryptosystem is a typical example of the mixed category in which the public key equations have mixed the variables of the plaintext and the variables of the ciphertext. Moreover, the public key equations have a total degree three with some simplifications in the secret polynomial. Patarin generated two versions of the Dragon family. The first was called Little Dragon and the second was called Big Dragon. In [133] and its extended version, Patarin proved that all these versions of Dragon as well as MIIP-3 are very efficient but insecure. The main conclusion of Patarin is that there is not an easy way to hide a monomial in order to avoid polynomial attack. As a result of this conclusion, Patarin suggested a candidate Dragon signature algorithm that was based on replacing a monomial with a complicated function. This Dragon signature was also recently broken in [186].

Recently, in the International Journal of Network Security & Its Applications (IJNSA), Singh et al. presented a new MPKC encryption scheme which is called Little Dragon Two (LD2 for short) that is constructed using permutation polynomials over finite fields [157]. According to the authors, LD2 is as efficient as Patarin's Little Dragon but secure against all the known attacks. Shortly after the publication appeared, linearization equations were found by Lei Hu which became known in the private communication with the first author of [157].

Due to these linearization equations, the authors of the LD2 scheme presented another scheme called Poly-Dragon [142]. Poly-Dragon as well as LD2 is constructed using permutation polynomials over finite fields. It is considered as an improved version of Patarin's Big Dragon cryptosystem. The Poly-Dragon scheme was also proposed as an efficient and secure scheme. In particular, the inventors of the Poly-Dragon scheme claim that Poly-Dragon resist algebraic cryptanalysis.

In this thesis we are interested in the HFE, the Little Dragon Two, and the Poly-Dragon cryptosystems, see Figure 2.2 for the relation and dependency for these cryptosystems. We use some instances from these cryptosystems to present a real world test cases for our suggested tools

for solving systems of multivariate quadratic polynomial equations. In the next subsections, we present the HFE cryptosystem. Afterwards, an overview of the Little Dragon Two cryptosystem is introduced followed by a description to the Poly-Dragon cryptosystem.

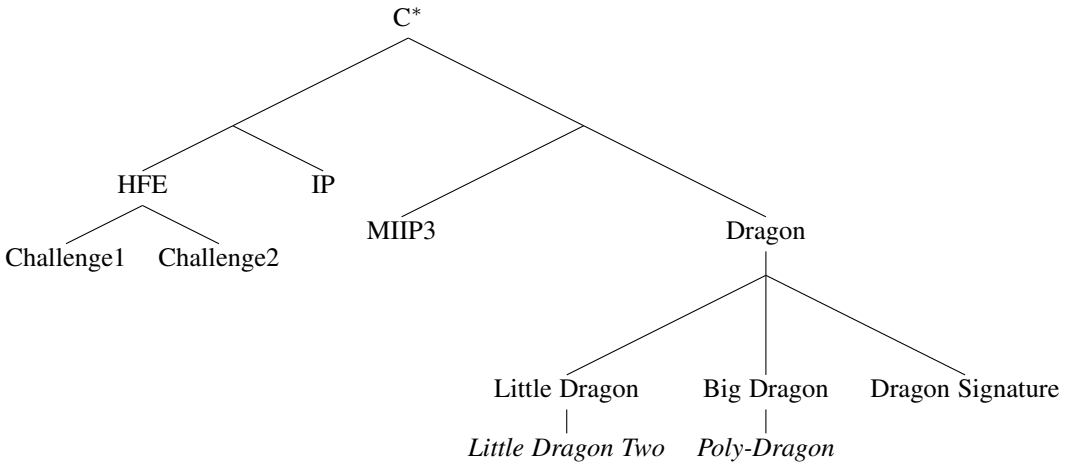

Figure 2.2: C*-based Cryptosystems

2.3.1 The HFE Cryptosystem

At Eurocrypt'96 paper [134], Patarin proposed two families of asymmetric algorithms namely HFE and IP. These two families were an improvement of the broken, by Patarin himself in [132], Matsumoto-Imai cryptosystem which was proposed at Eurocrypt'88 [117]. The Hidden Field Equations (HFE) is a class of asymmetric algorithms that were proposed to be used for encryption or signatures as well as authentication in an asymmetric way, the use of two related keys known as public and private keys such that it is computationally infeasible to derive the private key from the given public one. Furthermore, HFE can be used in order to give a very short asymmetric signatures or encrypted messages. The security of this scheme is not proved while this security is related to the problem of solving a system of multivariate quadratic polynomial equations over finite fields, MQ-problem.

The basic idea of the HFE scheme is to transform a univariate polynomial of a special form in an extension field into a system of multivariate quadratic polynomials over the underlying base finite field. The private key is represented as a univariate polynomial over the extension field and the system of multivariate quadratic polynomials is the representation of the public key. Beside this transformation, a two private affine transformations are used to hide the extension field and the private polynomial.

Let $\mathbb{F} = \mathbb{F}_q$ be the finite field with q elements, practically $q = 2$. Let $\mathbb{E}$ be the extension field of degree n over $\mathbb{F}$. The private polynomial, P' in X over $\mathbb{E}$, that was proposed in the basic HFE scheme has the so-called D-O polynomial shape [50]:

$$P'(X) = \sum_{0 \leq i \leq j \leq n} a_{ij} X^{q^i + q^j} + \sum_{0 \leq k \leq n} b_k X^{q^k} + c$$

with the a_{ij}, b_k, and $c \in \mathbb{E}$.

The degree of P' denoted by d is restricted such that P' can be easily evaluated and inverted quite efficiently. These restrictions can be achieved by applying some constraints on the powers of X such that all monomials of P' should be either of degree $q^i + q^j$, of degree q^k, or constant. The purpose of these restrictions is to keep the degree of public key polynomials as small as possible, particularly at most quadratic. In [134], Patarin suggested that $q \leq 64$, $n \leq 1024$, and $d \leq 1024$ as well as a theorem and its proof for the inversion of P'.

In order to hide the structure of P' and prevent its direct inversion, two affine transformations S and T are introduced. Each of the two affine transformations S and T can be represented by an invertible $n \times n$ matrix with elements from $\mathbb{F}$ and a column vector over $\mathbb{F}$ of length n. Thus for example, $S \in Aff(\mathbb{F})$, the set of affine transformations $\mathbb{F}^n \rightarrow \mathbb{F}^n$, can be written as $S(X) = M_s X^{Tr} + v_s$ where $M_s \in \mathbb{F}^{n \times n}$, $v_s \in \mathbb{F}^n$ and $X = (x_1, \ldots, x_n)$.

The extension field $\mathbb{E}$ may be represented as n dimensional vector space over $\mathbb{F}$ and any univariate polynomial $P' \in \mathbb{E}$ may be viewed as a collection of n multivariate polynomials in n variables $x_1, \ldots, x_n$ over $\mathbb{F}$, since $\mathbb{E}$ is isomorphic to $\mathbb{F}[z]/(f(z))$ where $f(z) \in \mathbb{F}[z]$ is irreducible polynomial of degree n. Let ϕ be the standard $\mathbb{F}$-linear map that identifies $\mathbb{E}$ with n-dimensional vector space $\mathbb{F}^n$, i.e., $\phi : \mathbb{F}^n \rightarrow \mathbb{E}$, defined by

$$\phi(a_1, a_2, \ldots, a_n) = a_1 + a_2 z + \cdots + a_n z^{n-1} \mod f(z)$$

Therefore, the private key of the HFE scheme consists of the two affine transformations S, T and the private polynomial P'. The public key, P, is obtained by combining the triple (T, P', S) as follows:

$$P = T \circ P' \circ S$$

On the assumption that some redundancy has been included in the representation of the plaintext, a message $X = (x_1, \ldots, x_n) \in \mathbb{F}^n$ is encrypted by applying Algorithm 2.1 and decryption is accomplished by using Algorithm 2.2. A discussion of the usage of HFE in signature or authentication can be found in [134] as well as its extended version.

Algorithm 2.1 HFE Encryption

1: **Inputs**
2: A plaintext message $X = (x_1, x_2, \ldots, x_n)$ of length n.
3: **Output**
4: A ciphertext message $Y = (y_1, y_2, \ldots, y_n)$ of length n.
5: **Begin**
6: Add redundancy r using a nonlinear function, e.g., hash, error correcting code.
7: Apply affine transformation S to X and get X', $X' = \phi \circ S(X)$.
8: Apply private polynomial P' to X' and obtain Y' = $P'(X')$.
9: Applying affine transformation T to Y' yields Y, $Y = T \circ \phi^{-1}(Y')$.
10: **End**

In the extended version of [134], Patarin proposed two explicit quadratic HFE challenge signature schemes. The first challenge, HFE(d=96, n=80), is a scheme that gives signatures of length 80 bits using a secret polynomial of degree 96 over $\mathbb{F}_2$. The second challenge, HFE(d=4352, n=144),

Algorithm 2.2 HFE Decryption

Inputs
A ciphertext message $Y = (y_1, y_2, \ldots, y_n)$ of length n and the secret parameters (S, T, P').
Output
A plaintext message $X = (x_1, x_2, \ldots, x_n)$ of length n.
Begin
Find all solutions z to the univariate equation $P'(z) = T^{-1}Y$.
Compute all $S^{-1}z$.
Use the redundancy to find X from all the solutions of the previous step.
End

gives signatures of length 144 bits. The hidden polynomial has a degree 4352 over $\mathbb{F}_{16}$ with 36 variables and 4 of the 36 equations are not given public. In [75], Faugère and Joux proposed a theoretical attack with a good complexity as well as a very practical method for breaking the first HFE challenge.

Several techniques for cryptanalysis of HFE were introduced after the publication of the HFE paper and its extended version. We will next review the related work history of HFE starting with older times ending with recent progress.

At Crypto'99, Kipnis and Shamir [106] presented a structural attack on a restricted version of HFE in which the private polynomial P' has not more than linear or constant terms and the two affine transformations S and T have only matrices part, no longer vectors. The key point of this attack is to reduce the problem of retrieving the private key from the public key to a problem of solving a system of ϵn^2 equations in n variables over the extension field using the Relinearization technique, see Chapter 3 for more details.

At CT-RSA'01, Nicolas Courtois [36] presented an improvement for the Kipnis-Shamir attack as well as an evaluation of the complexity of this attack. Moreover, Courtios designed and implemented two attacks, the Reconciliation attack and the Distillation attack. He also stated that they are much more efficient than the Kipnis-Shamir attack. The basic idea of the attack is to express the security of HFE in terms of properties of implicit equations that relate the inputs and the outputs of the encryption function in order to inverse this function without trying to recover the private key. The main contributions of these attacks are that the HFE challenge-1 can be broken in about 2^{62} and HFE can be considered secure for $d >$128, $n >$80.

In 2002, Christopher Wolf [175], presented his Diplomarbeit Thesis. In this thesis, a detailed description for the HFE scheme, attacks, variations, and applications can be found. In [178], Christopher Wolf et al. introduced a Java implementation of the HFE scheme as well as experimental data for its performance.

In 2003, Faugère and Joux [75] presented an efficient attack on the basic HFE cryptosystem based on Gröbner basis computations. The first main contribution of this attack was that the ability to break the HFE Challenge-1 in 96 hours on an HP workstation with an alpha EV68 processor at 1000Mhz. The second one was the ability to predict the maximal degree occurring in the Gröbner basis computations. Moreover, due to the algebraic properties of the secret key, when the degree of the secret polynomial is fixed, the cryptanalysis of an HFE system requires a polynomial time in the number of variables. The same results were represented in [94]. Based on the fact that an attacker knows the public key, the base field, and the degree of the extension

field, Ilia Toli [166] was able to find a single univariate polynomial, called "an alias of the public key" that is equivalent to the private polynomial. Therefore, the task of the attacker is reduced to the task of solving a single univariate polynomial equation. In [118], Michon et al. introduced an attempt to cryptanalize HFE based on BDD (Binary Decision Diagram). The main idea is to recover the plaintext from the given ciphertext and the public key which consists in satisfying a boolean formula.

In 2004, Courtios [42] presented three different algebraic attacks over $\mathbb{F}_{2^k}$ with applications to Sflach-v2, a multivariate-based signature scheme, and the HFE challenge-2. These three attacks were based on XLF, XL', and a modified FXL algorithms. As a conclusion that is related to HFE challenge-2, Courtios claimed that the best attack of HFE challenge-2 gives 2^{63}. In [179], Christopher Wolf and Bart Prenell outlined HFE, its variations (HFE$_-$, HFE$_v$), and the most recent attacks against HFE at that time. Moreover, they described the signature scheme Quartz and sketched two versions of it which were immune against these attacks.

In 2005, Jintai Ding and Dieter Schmidt [60] presented an algebraic attack on the HFE$_v$. Based on the structure of the HFE$_v$, the new Vinegar variables can be seen as an external perturbation then the attacker can try to separate them. Moreover, Jintai Ding and Dieter Schmidt used the method of internal perturbation developed by Ding [54] to present a new signature scheme called IPHFE that immunes the new suggested attack. In [91], Omessaad Hamdi et al. described a generalization of Kipnis and Shamir attack which is based on Fourier Transformation and is applicable to HFE instances of degree more than two.

In 2006, Louis Granboulan et al. studied the complexity of the decryption attack which uses Gröbner basis to recover the plaintext given the ciphertext and the public key in [89]. The main result of this work is that the decryption attack has a sub-exponential complexity that is much smaller than the classical sub-exponential complexity encountered in factoring or discrete logarithm computations. They called this complexity as Quasipolynomial. In [92], Omessaad Hamdi et al. outlined the advantages and disadvantages of HFE and studied the effect of the extension field degree and the private polynomial degree on the performance of HFE. Based on the differential properties of the public key in HFE, Vivien Dubois et al. presented a distinguisher for HFE that can distinguish between the HFE public key and random systems of quadratic polynomials in [66].

In 2007, under the title "HFE solved or saved?", Fabian Werner [173] presented his B.Sc. thesis. The main idea is to try to solve HFE systems that are saved from the algebraic attack. Fabian Werner followed the direction to save HFE from algebraic attacks by giving HFE two adaptation methods. The first one is to adapt the characteristics of the base field and the second method is to use internal perturbation. The main conclusion of that thesis is that over finite field having a sufficiently large characteristic, HFE can defeat the algebraic attacks.

In 2008, Kipnis-Shamir attack on HFE was re-examined in [97]. The main contribution of this paper is that, not only the claims in the original Kipnis and Shamir's attack on the HFE cryptosystem are invalid but also the improved attack by Courtios. Moreover, Kipnis and Shamir's attack should be exponential not polynomial. This was showed by means of practical experiments as well as theoretical arguments. Jintai Ding et al. studied the effect of using algebraic attacks on the HFE over a finite field whose characteristic is not two in [63] and presented a new HFE variant over a finite field of odd characteristic with an extra embedding modifier in [29].

In 2009, Dhananjoy Dey et al. [51] designed a new hash function called HF-hash. The compression function of HF-hash was designed by using the first 32 polynomials of HFE challenge-1

with 64 variables by forcing remaining 16 variables as zero.

In 2010, Charles Bouillaguet et al. presented an attack to a family of weak keys of HFE instances in [18]. In these instances the secret polynomial is defined over the base field rather than the extension field. In this context, the problem of key recovering is reduced to an IP problem. Therefore, the solution of this problem allows the attacker to recover an equivalents to the secret elements.

2.3.2 LD2: Little Dragon Two Cryptosystem

The Little Dragon Two, LD2, multivariate public-key cryptosystem is a mixed type scheme that has a public key in which plaintext and ciphertext variables are "mixed" together. LD2 is a modified version of Patarin's Little Dragon cryptosystem and it is constructed using permutation polynomials over finite fields. In this section, we present an overview of the LD2 scheme. In MPKCs, the main security parameters are the number of equations and the number of variables. The authors of LD2 did not propose any specific parameters. For a more detailed explanation see [157].

Definition 2.1. *Let $\mathbb{F}_q$ be the finite field of $q = p^n$ elements where p is prime and n is a positive integer. A polynomial $f \in \mathbb{F}_q[x_1, \ldots, x_n]$ is called a permutation polynomial in n variables over $\mathbb{F}_q$ if and only if one of the following equivalent conditions holds:*

1. *the function f is onto.*
2. *the function f is one-to-one.*
3. *$f(x) = a$ has a solution in $\mathbb{F}_q$ for each $a \in \mathbb{F}_q$.*
4. *$f(x) = a$ has a unique solution in $\mathbb{F}_q$ for each $a \in \mathbb{F}_q$.*

Simply speaking, this means that a polynomial $f \in \mathbb{F}_q[x_1, \ldots, x_n]$ is a permutation polynomial over $\mathbb{F}_q$ if it induces a bijective map from $\mathbb{F}_q$ to itself.

Lemma 2.2. *Let $Tr(x)$ denotes the trace function on the field $\mathbb{F}_{2^n}$ i.e. $Tr : \mathbb{F}_{2^n} \to \mathbb{F}_2$ is defined by $Tr(x) = x + x^2 + x^{2^2} + \cdots + x^{2^{n-1}}$. The polynomial $g(x) = (x^{2^{rk}} + x^{2^r} + \alpha)^\ell + x$ is a permutation polynomial of $\mathbb{F}_{2^n}$, when $Tr(\alpha) = 1$ and $\ell \cdot (2^{2^{rk}} + 2^r) \equiv 1 \pmod{2^n - 1}$.*

A proof of Lemma 2.2 is presented by the authors of [157]. A more detailed explanation for permutation polynomials and trace representation on finite fields can be found in [112].

Suppose that $X = (x_1, \ldots, x_n)$ denotes the plaintext variables and $Y = (y_1, \ldots, y_n)$ denotes the ciphertext variables. In the LD2 scheme, the public key equations are multivariate polynomials over $\mathbb{F}_2$ of the form:

$$\begin{cases} p_1(x_1, x_2, \ldots, x_n,\ y_1, y_2, \ldots, y_n) = 0, \\ p_2(x_1, x_2, \ldots, x_n,\ y_1, y_2, \ldots, y_n) = 0, \\ \vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ p_\lambda(x_1, x_2, \ldots, x_n,\ y_1, y_2, \ldots, y_n) = 0, \end{cases}$$

where $p_1, p_2, \ldots, p_\lambda \in \mathbb{F}_2[x_1, \ldots, x_n,\ y_1, \ldots, y_n]$ are polynomials of total degree two.

Due to the restrictions on r, k and ℓ placed by Lemma 2.2, there are a few choices for r, k and ℓ to produce a permutation polynomial $g(x) = (x^{2^{rk}} + x^{2^r} + \alpha)^\ell + x$ and to use $g(x)$ to design a public-key scheme with a quadratic public key. We can choose $r = 0, n = 2m - 1, k = m$ and $\ell = 2^m - 1$ for example, then $G(x) = (x^{2^m} + x + \alpha)^{2^m-1} + x$ is a permutation polynomial. By choosing $r = 0, n = 2m - 1, k = m$ and $\ell = 2^m + 1$, the authors of [157] stated that it is not clear whether $G'(x) = (x^{2^m} + x + \alpha)^{2^m+1} + x$ is a permutation polynomial or not while it can produce a quadratic public key.

Let S and T be two invertible affine transformations. Then the plaintext can be permuted to a ciphertext using the relation $G(S(x_1, \ldots, x_n)) = T(y_1, \ldots, y_n)$. Suppose $S(x_1, \ldots, x_n) = u$ and $T(y_1, \ldots, y_n) = v$ therefore, the relation between plaintext and ciphertext can be written as follows:

$$\begin{aligned}
(u^{2^m} + u + \alpha)^{2^m-1} + u &= v \\
(u^{2^m} + u + \alpha)^{2^m-1} + (u + v) &= 0 \\
(u^{2^m} + u + \alpha)^{2^m} + (u + v)(u^{2^m} + u + \alpha) &= 0 \\
((u^{2^m} + u) + \alpha)^{2^m} + u^{2^m+1} + u^2 + u\alpha + vu^{2^m} + vu + v\alpha &= 0 \\
(u^{2^m} + u)^{2^m} + \alpha^{2^m} + u^{2^m+1} + u^2 + u\alpha + vu^{2^m} + vu + v\alpha &= 0 \\
&\vdots \\
u^{2^m} + \alpha^{2^m} + u^{2^m+1} + u\alpha + vu^{2^m} + vu + v\alpha &= 0 \\
u^{2^m+1} + u^{2^m}v + uv + u\alpha + u^{2^m} + v\alpha + \alpha^{2^m} &= 0
\end{aligned} \tag{2.1}$$

It is known that the extension field $\mathbb{F}_{2^n}$ can be viewed as a vector space over $\mathbb{F}_2$. Let $\beta = \{\beta_1, \beta_2, \ldots, \beta_n, \}$ be a normal basis of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$ for some $\beta \in \mathbb{F}_{2^n}$. Therefore any $z \in \mathbb{F}_{2^n}$ can be expressed as $z = \sum_{i=1}^{n} z_i\beta_i$, where $z \in \mathbb{F}_2$. By substituting $u = S(x_1, \ldots, x_n)$ and $v = T(y_1, \ldots, y_n)$, Equation (2.1) can be represented as n quadratic polynomial equations of the form:

$$\sum a_{ij}x_ix_j + \sum b_{ij}x_iy_j + \sum c_ky_k + \sum d_kx_k + e_l = 0 \tag{2.2}$$

where the coefficients $a_{ij}, b_{ij}, c_k, d_k, e_l \in \mathbb{F}_2$.

In Equation (2.2), the terms of the form $\sum x_ix_j + \sum x_k + c_1$ are obtained from u^{2^m+1}, the terms of the form $\sum x_iy_j + \sum x_k + \sum y_k + c_2$ are obtained from $u^{2^m}v + uv$, the terms of the form $\sum x_i + c_3$ are obtained from $u\alpha + u^{2^m}$ and $v\alpha$ gives the terms of the form $\sum y_i + c_4$, where c_1, c_2, c_3 and c_4 are constants.

The secret parameters are the finite field element α and the two invertible affine transformations S and T. A plaintext $X = (x_1, x_2, \ldots, x_n) \in \mathbb{F}_2^n$ is encrypted by applying Algorithm 2.3. Decryption is accomplished by using Algorithm 2.4.

A discussion for the security and the efficiency of the proposed scheme is presented in [157]. As a conclusion, the authors claimed that they present an efficient and secure multivariate public key cryptosystem that can be used for both encryption and signatures.

Algorithm 2.3 LD2 Encryption

Inputs
  A plaintext message $X = (x_1, x_2, \ldots, x_n)$ of length n.
Output
  A ciphertext message $Y = (y_1, y_2, \ldots, y_n)$ of length n.
Begin
Substitute the plaintext $(x_1, x_2, \ldots, x_n)$ in the public key.
Get n linear equations in the ciphertext variables $(y_1, y_2, \ldots, y_n)$.
Solve these linear equations by Gaussian elimination method to obtain the correct ciphertext $Y = (y_1, y_2, \ldots, y_n)$.
End

Algorithm 2.4 LD2 Decryption

Inputs
  A ciphertext message $Y = (y_1, y_2, \ldots, y_n)$ of length n and the secret parameters (S, T, α).
Output
  A plaintext message $X = (x_1, x_2, \ldots, x_n)$ of length n.
Begin
Let $v = T(y_1, y_2, \ldots, y_n)$.
Let $z_1 = \alpha + 1 + v + v^{2^m}$.
Let $z_2 = z_1^{2^m-1}$.
Let $z_3 = v + 1 + z_2$.
Let $X_1 = S^{-1}(v + 1)$.
Let $X_2 = S^{-1}(z_3)$.
Return (X_1, X_2), Either X_1 or X_2 is the required secret message.
End

2.3.3 Poly-Dragon Cryptosystem

The Poly-Dragon multivariate-based public-key cryptosystem is a mixed type scheme that has a public key of total degree three, two in plaintext and one in ciphertext. Poly-Dragon is based on permutation polynomials and is supposed to be as efficient as Patarin's Big Dragon [133]. As well as LD2, Poly-Dragon did not have any proposed security parameters. In this section, we introduce an overview of the Poly-Dragon scheme. See [142, 158] for more details.

Definition 2.3. *Let $\mathbb{F}_q$ be the finite field of characteristic p. A polynomial of the form*

$$L(x) = \sum_i \alpha_i x^{p^i}$$

with coefficients in an extension field $\mathbb{F}_q$ of $\mathbb{F}_p$ is called a p-polynomial over $\mathbb{F}_q$.

Simply speaking, a polynomial over $\mathbb{F}_q$ is said to be a p-polynomial over $\mathbb{F}_q$ if each of its terms has a degree equal to a power of p. A p-polynomial is also called linearized polynomial because for all $\beta, \gamma \in \mathbb{F}_q$ and $a \in \mathbb{F}_p$ it satisfies the following properties: $L(\beta + \gamma) = L(\beta) + L(\gamma)$ and $L(a\beta) = aL(\beta)$. In [112], it is proved that $L(x)$ is a permutation polynomial of $\mathbb{F}_q$ if and only if the only root of $L(x)$ in $\mathbb{F}_q$ is 0.

Proposition 2.4. *Let $L_\beta(x) = \sum_{i=0}^{n-1} \beta_i x^{2^i} \in \mathbb{F}_{2^n}$ be a p-polynomial defined with n an odd positive integer and $\beta = (\beta_1, \beta_2, \ldots, \beta_n) \in \mathbb{F}_{2^n}$ such that the weight of β is even and that 0 and 1 are the only roots of $L_\beta(x)$. Then*

$$f(x) = (L_\beta(x) + \gamma)^\ell + Tr(x)$$

is a permutation polynomial of $\mathbb{F}_{2^n}$, where l is any positive integer with $(2^{k_1} + 2^{k_2}) \cdot \ell \equiv 1 \pmod{2^n - 1}, \gamma \in \mathbb{F}_{2^n}$ with $Tr(\gamma) = 1$ and k_1 , k_2 are non negative integers such that $gcd(2^{k_1} + 2^{k_2}, 2^n - 1) = 1$.

Proposition 2.5. *The polynomial $g(x) = (x^{2^{k2^r}} + x^{2^r} + \alpha)^\ell + x$ is a permutation polynomial of $\mathbb{F}_{2^n}$ if $Tr(\alpha) = 1$ and $(2^{k2^r} + 2^r) \cdot \ell \equiv 1 \pmod{2^n - 1}$.*

The two permutation polynomials $g(x) = (x^{2^{k2^r}} + x^{2^r} + \alpha)^\ell + x$ and $f(x) = (L_\beta(x) + \gamma)^\ell + Tr(x)$ from Proposition 2.4 and Proposition 2.5 are used in Poly-Dragon public-key cryptosystem. The permutation polynomials in which ℓ is of the form $2^m - 1$ and $r = 0, n = 2m - 1, k = m, k_2 = m$ and $k_1 = 0$ are used to generate the public key. Therefore, for key generation $G(x) = (x^{2^m} + x + \alpha)^{2^m - 1} + x$ and $F(x) = (L_\beta(x) + \gamma)^{2^m - 1} + Tr(x)$ are used where α, β, γ are secret.

The relation between plaintext $X = (x_1, x_2, \ldots, x_n)$ and ciphertext $Y = (y_1, y_2, \ldots, y_n)$ can be written as $G(S(x_1, x_2, \ldots, x_n)) = F(T(y_1, y_2, \ldots, y_n))$, where S and T are two invertible affine transformations. This relation can be written as $(u^{2^m} + u + \alpha)^{2^m - 1} + u = (L_\beta(v) + \gamma)^{2^m - 1} + Tr(v)$ such that $S(x_1, x_2, \ldots, x_n) = u$ and $T(y_1, y_2, \ldots, y_n) = v$. Multiplying by $(u^{2^m} + u + \alpha) \times (L_\beta(v) + \gamma)$, which is a nonzero in the field $\mathbb{F}_{2^n}$, we obtain:

$$\begin{aligned} &(u^{2^m} + u + \alpha)^{2^m}(L_\beta(v) + \gamma) + u(u^{2^m + u + \alpha})(L_\beta(v) + \gamma) \\ &+(u^{2^m} + u + \alpha)(L_\beta(v) + \gamma)^{2^m} + Tr(v)(u^{2^m + u + \alpha})(L_\beta(v) + \gamma) = 0 \end{aligned} \tag{2.3}$$

The extension field $\mathbb{F}_{2^n}$ can be viewed as a vector space over $\mathbb{F}_2$. Then we can identify $\mathbb{F}_{2^n}$ with $\mathbb{F}_2^n$. Let $Tr(v) = \zeta_y \in \{0,1\}$ and by substituting $u = S(x_1, x_2, \ldots, x_n)$ and $v = T(y_1, y_2, \ldots, y_n)$, in Equation (2.3), we obtain n non-linear polynomials equations of degree three of the form:

$$\sum a_{ijk}x_ix_jy_k + \sum b_{ij}x_ix_j + \sum (c_{ij}+\zeta_y)x_iy_j + \sum (d_k+\zeta_y)y_k + \sum (e_k+\zeta_y)x_k + f_l, \quad (2.4)$$

where $a_{ijk}, b_{ij}, c_{ij}, d_k, e_k, f_l \in \mathbb{F}_2$.

The secret parameters are the finite field elements α, β, γ and the two invertible affine transformations S and T. A plaintext $X = (x_1, x_2, \ldots, x_n) \in \mathbb{F}_2^n$ is encrypted by applying Algorithm 2.5. Decryption is accomplished by using Algorithm 2.6.

In [142], the authors stated a proof for the validity of the generated plaintext by the decryption algorithm. A discussion for the security and the efficiency of the proposed scheme is also presented. As a conclusion, the authors claimed that they presented an efficient and secure multivariate public key cryptosystem that can be used for encryption as well as for signature.

Algorithm 2.5 Poly-Dragon Encryption

1: **Inputs**
2: A plaintext message $X = (x_1, x_2, \ldots, x_n)$ of length n.
3: **Output**
4: A ciphertext message pair $(Y^{'}, Y^{''})$.
5: **Begin**
6: Substitute the plaintext variables $(x_1, x_2, \ldots, x_n)$ and $\zeta_y = 0$ in the public key.
7: Get n linear equations in the ciphertext variables $(y_1, y_2, \ldots, y_n)$.
8: Solve these linear equations by Gaussian elimination method to obtain the ciphertext variables $Y^{'} = (y_1, y_2, \ldots, y_n)$.
9: Substitute the plaintext $(x_1, x_2, \ldots, x_n)$ and $\zeta_y = 1$ in the public key.
10: Get n linear equations in the ciphertext $(y_1, y_2, \ldots, y_n)$.
11: Solve these linear equations by Gaussian elimination method to obtain the ciphertext variables $Y^{''} = (y_1, y_2, \ldots, y_n)$.
12: Return the ordered pair $(Y^{'}, Y^{''})$ as the required ciphertext.
13: **End**

2.4 Cryptanalysis of Multivariate-based Cryptosystems

Cryptanalysis is the study of methods for attempting to break cryptographic schemes. These methods include recovering the meaning (plaintext) of encrypted information (ciphertext) without the knowledge of the secret key, encrypting new information in lieu of the sender without access to the secret information (key) that is supposed to be the only way to do so and recovering the secret key. The goal of cryptanalysis is to find some weakness in a cryptosystem. Cryptanalysis, attack, or breaking the cryptosystem are three terms that are used to show such weakness or insecurity. The

Algorithm 2.6 Poly-Dragon Decryption

Inputs
A ciphertext message $(Y^{'}, Y^{''})$ and the secret parameters $(S, T, \alpha, \beta, \gamma)$.
Output
A plaintext message $X = (x_1, x_2, \ldots, x_n)$ of length n.
Begin
Let $v_1 = T(Y^{'})$ and $v_2 = T(Y^{''})$.
Let $z_1 = L_\beta(v_1) + \gamma$ and $z_2 = L_\beta(v_2) + \gamma$.
Let $\bar{z_3} = z_1^{2^m-1}$ and $\bar{z_4} = z_2^{2^m-1}$.
Let $z_3 = \bar{z_3} + Tr(v_1)$ and $z_4 = \bar{z_4} + Tr(v_2)$.
Let $z_5 = z_3^{2^m} + z_3 + \alpha + 1$ and $z_6 = z_4^{2^m} + z_4 + \alpha + 1$.
Let $z_7 = z_5^{2^m-1}$ and $z_8 = z_6^{2^m-1}$.
Let $X_1 = S^{-1}(z_3 + 1)$ and Let $X_2 = S^{-1}(z_4 + 1)$.
Let $X_3 = S^{-1}(z_3 + z_7 + 1)$ and Let $X_4 = S^{-1}(z_4 + z_8 + 1)$.
Return (X_1, X_2, X_3, X_4), Either X_1, X_2, X_3 or X_4 is the required secret message.
End

cryptanalyst's (or attacker's) aim is to improve cryptosystems while the hacker uses cryptanalysis with bad intentions.

From a historical point of view, the first known recorded explanation of cryptanalysis was given by 9th-century Arabian polymath, Al-Kindi (also known as "Alkindus" in Europe). in an article titled "A Manuscript on Deciphering Cryptographic Messages". In particular, he is credited with developing the frequency analysis method whereby variations in the frequency of the occurrence of letters could be analyzed and exploited to break ciphers [159].

Brute-force attack (also called exhaustive search) is a general theoretical approach to attack any cryptosystem. It is based on trying all the possible combinations of the secret information. In MPKCs context, the attacker task is to recover the plaintext from a given public key equations and a ciphertext. Therefore, the attacker can try all the possible combinations of the plaintext one by one and substitute in the public key equations until the right ciphertext is obtained. Indeed, the difficulty of a Brute-force attack depends on the number of variables in the systems, the number of equations and the base field elements. By anyhow, a Brute-force attack will always succeed. However, Brute-force attacks against systems with sufficiently number of variables and equations may require billions of years to complete. Therefore, a method that has a complexity less than that of the Brute-force attack reflects a weakness in the cryptosystem.

Practically, there are many methods to cryptanalysis multivariate-based cryptosystems. These methods can be classified as inversion attack and key recovery attack [177]. The inversion attack is based on recovering a plaintext for a given ciphertext and public key. While the key recovery attack aimed to recover the private key or an equivalent one for a given public key. Another classification way is specific attack and general attack [65]. Specific attacks are based on the internal structure of the cryptosystem. The general attack depends on the general methods of solving a set of multivariate polynomial equations. A survey of attacks on multivariate-based cryptosystems can be found in [77].

In cryptography, constructing a system of multivariate polynomial equations that describe a cryptographic primitive in which the secret is represented by unknown variables and then solving

this system to recover that secret information is called algebraic cryptanalysis (attack). These attacks belong to the general attack category with the aim of recovering the plaintext for a given ciphertext and public key (inversion attack).

Algebraic attacks are applicable to a variety of ciphers: block ciphers like AES [32] and Serpent [46], stream ciphers like Toycrypt [41] and E0 [7], and asymmetric cryptosystems like HFE [75]. An overview about algebraic cryptanalysis for different cryptosystems can be found in [15, 11].

Currently there are many different algorithms which are based on different strategies to solve systems of multivariate quadratic polynomial equations. Gröbner basis methods and linearization are the two main categories to solve such systems of equations. Gröbner basis based algorithms include: Buchberger [20], F4 [72] and F5 [73], while linearization based algorithms include: Relinearization [107] and XL [44]. Based on extended Dixon Resultants, another algorithm DR [163] is proposed to solve system of multivariate polynomial equations by considering $x_1, \ldots, x_{n-1}$ as variables and x_n as a parameter. Ding, Gower and Schmidt presented another algorithm called Zhuang-Zi [58] which is based on the usage of Frobenius maps. GeometricXL [128] is an algorithm that generalizes the XL algorithm to a geometrically invariant derivate. Another approach is to transform the system of equations into a Boolean satisfiability problem and solve using a SAT-solver [13]. See Chapter 3 for more details.

2.5 Conclusion

Failure, hope and work are the trinity of success. The failure to secure most multivariate-based public-key cryptosystems (MPKCs), the hope that these MPKCs will resistant quantum computing based attacks and the work (cryptanalysis) to improve these MPKCs will not end until the success to obtain a secure scheme is achieved. Until we get to this moment, obtaining a secure MPKC, the cryptographers will improve their schemes after applying a cryptanalysis to it and the cryptanalysts will improve their tools to discover a weakness or insecurity schemes. These processes will remain successive such as the sequential succession of night and day.

In this context, we are interested in improving current tools to cryptanalyze the MPKCs, particularly the XL algorithm which is one of the tools that is used from the algebraic cryptanalysis toolbox. In the next chapter a discussion is presented for the XL algorithm.

3 The XL Algorithm: State of the Art

The main target of this chapter is to review the XL algorithm by surveying it from creation time till nowadays. This is followed by experimental results of solving different dense random systems of the MQ-problem over $\mathbb{F}_2$. These experiments are based on the fastest algorithms in the M4RI package. The aim of these experiments is to provide a practical limit for the XL algorithm under our available resources, 128GB memory. Before ending the chapter, a discussion of the other solvers and their relation/comparison to XL is introduced.

3.1 Introduction

The problem of solving systems of polynomial equations over finite fields has several important applications to the theory and practice of many computational problems arising in sciences, engineering, business management, and statistics. This problem, at least obtaining one solution, is known to be one of the three major problems in the theory of polynomial equations over finite fields. The other two problems are the determination of a given system over a given finite field whether has a solution or not and the determination of the number of solutions to a known solvable system.

In this thesis, we are interested only in solving systems of multivariate quadratic polynomial equations over the finite field $\mathbb{F}_2$. We consider two categories of such systems namely dense random systems and systems that are arising from multivariate-based cryptosystems such as HFE, Little Dragon Two, and Poly-Dragon which are presented in Chapter 2.

In cryptography, the XL algorithm was presented as a simple and powerful algorithm for solving overdetermined systems of polynomial equations. The main idea of this algorithm is to produce from each original polynomial a large number of higher degree polynomials by multiplying the original polynomial with all possible monomials up to some bounded degree, then XL linearizes the extended system and solves it using Gaussian elimination. The main goal of this thesis is to improve this algorithm and test these improvements using applications to algebraic cryptanalysis. Before introducing these improvements, we want to present the maximum limits that we can reach with XL.

In this chapter, the Relinearization technique is briefly introduced and shown to be related to the XL algorithm. Next, XL is studied in more details and the complexity estimations by its inventors are presented. We present an overview of the different variants of XL and a variety of different techniques that are used to solve polynomial equations over finite fields. We hint possible connections or similarity to these techniques with XL. We discuss our implementation for XL and the results for solving dense random systems of multivariate quadratic polynomial equations over $\mathbb{F}_2$. At the end of the chapter we conclude XL.

3.2 The Relinearization Technique

The initial idea to solve a system of multivariate quadratic polynomial equations is to convert it to a linear system of equations. A well-known and familiar general technique for this conversion is the Linearization technique. The basic idea for Linearization is to replace every monomial as a new independent variable. The new linear system is known as the linearized system and can be easily solved with basic linear algebra. Any solution of the original system is also a solution of the new linearized system. The success of Linearization depends on the number of linearly independent polynomials in the linearized system. If the rank of the new linearized system is significantly less than the number of monomials in the original system, the new linearized system can produce far too many possible incorrect solutions to the original system. These solutions are so-called parasitic solutions. Therefore, the Linearization technique is applicable to a limited types of polynomial equations.

In the case that the Linearization technique fails, Relinearization sometimes can be used. The Relinearization technique was proposed and used to analyze the HFE scheme by Kipnis and Shamir in [107]. Furthermore, it is used to break the Double-Round quadratic cryptosystem in [145] and recently to analyze the BMQE (Bisectional Multivariate Quadratic Equations) system [187]. The idea behind Relinearization is to rewrite a system of $m = \epsilon n^2$ homogeneous quadratic equations in n variables $x_1, \ldots, x_n$ where ϵ is smaller than $1/2$, as a new system of m linear equations in the $n^2/2$ new variables $y_{ij} = x_i x_j$ then construct more equations using the commutative property of multiplication and connection between new variables, i.e., $y_{ij}y_{kl} = y_{ik}y_{il}$. The resulting system is solved either by another Linearization or by recursive Relinearization. The entire Relinearization algorithm is described in Algorithm 3.1 followed by an illustrative example from [107].

Algorithm 3.1 Relinearization

1: **Inputs**
2: A set of $m = \epsilon n^2$ quadratic polynomial equations in n variables.
3: **Output**
4: A solution of the input set of equations.
5: **Begin**
6: Regard each monomial $x_i x_j$, $i \leq j$ as a new variable $y_{ij} = x_i x_j$.
7: Solve the linearized system (e.g using Gaussian elimination). If there exist at least a value of one variable, substitute with that value in the equations and solve again.
8: If the system not solved, express every variable y_{ij} as a linear combination of the free variables z_k, where k is smaller than the total number of monomials up to degree two.
9: Create additional quadratic equations which express the commutativity of multiplication of $x_i x_j$. (i.e.) let $1 \leq i \leq j \leq k \leq \ell \leq n$ be any 4-tuple of indexes then
$(x_i x_j)(x_k x_\ell) = (x_i x_k)(x_j x_\ell) = (x_i x_\ell)(x_j x_k) \Rightarrow y_{ij}y_{k\ell} = y_{ik}y_{j\ell} = y_{i\ell}y_{jk}$
10: Express each y_{ij} as a linear combination of z_k. The resulting is a quadratic in z_k variables.
11: Regard each monomial $z_i z_j$, $i \leq j$ as a new variable $w_{ij} = z_i z_j$, (i.e. Relinearize)
12: Solve the linearized system.
13: Find the solution of the original system by back substitution in the different variables w_{ij}, z_k, y_{ij}, x_i

Example 3.1. *Suppose we want to solve the following system of polynomial equations in* $\mathbb{F}_7[x_1, x_2, x_2]$.

$$\begin{aligned} 3x_1^2 + 5x_1x_2 + 5x_1x_3 + 2x_2^2 + 6x_2x_3 + 4x_3^2 &= 5 \\ 6x_1^2 + x_1x_2 + 4x_1x_3 + 4x_2^2 + 5x_2x_3 + x_3^2 &= 6 \\ 5x_1^2 + 2x_1x_2 + 6x_1x_3 + 2x_2^2 + 3x_2x_3 + 2x_3^2 &= 5 \\ 2x_1^2 + x_1x_3 + 6x_2^2 + 5x_2x_3 + 5x_3^2 &= 0 \\ 4x_1^2 + 6x_1x_2 + 2x_1x_3 + 5x_2^2 + x_2x_3 + 4x_3^2 &= 0 \end{aligned}$$

Step *1: For every product* x_ix_j*, introduce a new variable* y_{ij}*. (Line 6 in Algorithm3.1)*

$$\begin{bmatrix} y_{11} & y_{12} & y_{13} & y_{22} & y_{23} & y_{33} & c \\ 3 & 5 & 5 & 2 & 6 & 4 & 5 \\ 6 & 1 & 4 & 4 & 5 & 1 & 6 \\ 5 & 2 & 6 & 2 & 3 & 2 & 5 \\ 2 & 0 & 1 & 6 & 5 & 5 & 0 \\ 4 & 6 & 2 & 5 & 1 & 4 & 0 \end{bmatrix}$$

Step *2: Perform Gaussian elimination to the linearized system of equations. (Line 7 in Algorithm3.1)*

$$\begin{bmatrix} y_{11} & y_{12} & y_{13} & y_{22} & y_{23} & y_{33} & c \\ 1 & 2 & 0 & 0 & 0 & 0 & 2 \\ 0 & 5 & 1 & 0 & 0 & 0 & 3 \\ 0 & 3 & 0 & 1 & 0 & 0 & 6 \\ 0 & 6 & 0 & 0 & 1 & 0 & 6 \\ 0 & 4 & 0 & 0 & 0 & 1 & 5 \end{bmatrix}$$

Step *3: Express each* y_{ij} *as a linear combination of* $k < \frac{n.(n+1)}{2}$ *new parameters* $z_1, \ldots, z_k$*. (Line 8 in Algorithm3.1)*
Here $k = 1$*, simply we use* z *instead of* z_1*.* $y_{11} = 2z + 2$*,* $y_{12} = z$*,* $y_{13} = 5z + 3$*,* $y_{22} = 3z + 6$*,* $y_{23} = 6z + 6$ *and* $y_{33} = 4z + 5$ *with* $z \in \mathbb{F}_7$*.*

Step *4: Create additional equations which express the commutativity of the multiplication of* x_ix_j*. (Line 9 in Algorithm3.1)*
$y_{11}y_{23} = y_{12}y_{13}$*,* $y_{12}y_{23} = y_{13}y_{22}$*,* $y_{12}y_{33} = y_{13}y_{23}$

Step *5: Express the new equations using the* z_k*'s. (Line 10 in Algorithm3.1)*
$y_{11}y_{23} = y_{12}y_{13} \Longrightarrow 3z^2 + 6z + 5 = 0$
$y_{12}y_{23} = y_{13}y_{22} \Longrightarrow 2z^2 + 1z + 4 = 0$
$y_{12}y_{33} = y_{13}y_{23} \Longrightarrow 6z^2 + 4z + 4 = 0$

Step *6: we apply a* **relinearization** *step. (Line 11 in Algorithm3.1)*
Introduce $w_1 = z$ *and* $w_2 = z^2$

Step *7: Solve the new system by linearization again or perhaps recursively with regularization. (Line 12 in Algorithm3.1)*
find $w_1 = 1$ *and* $w_2 = 6$*.*

Step *8: Find the original solution by using z. (Line 13 in Algorithm3.1)*
$y_{11} = 4$, $y_{22} = 2$ *and* $y_{33} = 2$ *and hence* $x1 = \{2, 5\}$, $x2 = \{3, 4\}$ *and* $x3 = \{3, 4\}$. *Finally,* $y_{12} = 6$ *and* $y_{23} = 5$ *imply* $(x_1, x_2, x_3 \in \{(2, 3, 4), (5, 4, 3)\})$.

In the above algorithm, all the new created equations that are based on the commutativity of the multiplication of $x_i x_j$ are of degree 4 in the original variables. So, this method is referred to as degree 4 Relinearization. There are higher degree variants of Relinearization, in which we could add equations with higher degrees. In degree 6 Relinearization for example, we could add equations of the form $y_{ij}y_{kl}y_{mn} = y_{ik}y_{jm}y_{ln}$.

In [121], T. Moh analyzed the theoretical aspects of the Relinearization technique and showed that Relinearization is rather impractical for the Tame Transformation Method (TTM) scheme. Moreover, in [44] the authors show that many of the equations added by higher degree Relinearization are linearly dependent. Thus the algorithm is less efficient than initially claimed and the types of polynomial equations which can be successfully solved by the technique are limited. The key point, then, is to generate more linearly independent equations. This was the main idea for the so-called XL algorithm.

3.3 The XL Algorithm

The XL algorithm, stands for eXtended Linearization, was proposed in [44] as a simple and powerful algorithm for solving overdetermined systems of polynomial equations. The general strategy of XL can be viewed as a combination of bounded degree Gröbner basis and linearization [8]. The main idea of this algorithm is to produce from each original polynomial a large number of higher degree polynomials by multiplying the original polynomial with all possible monomials up to some bounded degree, then XL linearizes the extended system to be solved using Gaussian elimination. XL was designed to solve a system of multivariate polynomial equations that has only one solution over a finite field. This assumption of uniqueness solution is implicitly noted in [44] and explicitly stated in [45].

In order to define the algorithm itself, we introduce some notations from [44]. This followed by an example that demonstrate the different steps of XL.

Let K be a field. We represent the number of variables by n and the number of equations by m. Let $P = \{p_1, \ldots, p_m\}$ be a set of multivariate quadratic polynomials such that $p_i \in K[x_1, \ldots, x_n]$, $1 \leq i \leq m$. Let $x^k = \{x_1^{k_1} \cdot x_2^{k_2} \ldots x_n^{k_n} : k_i \in K, k_1 + \cdots + k_n = k\}$ be the set of all monomials that have degree k, denoted by $\prod_{j=1}^{k} x_{i_j}$ with all the i_j being pairwise different. The main idea of the XL algorithm is to solve $p_i(x_1, \ldots, x_n) = 0$ by the linearization of the system of all polynomial equations $\prod_{j=1}^{k} x_{i_j} \cdot p_i(x_1, \ldots, x_n) = 0$, $k \leq D - 2$. Let $I_D \subset I$ be the linear space generated by all the equations $\prod_{j=1}^{k} x_{i_j} \cdot p_i(x_1, \ldots, x_n) = 0$, $k \leq D - 2$ and I be the ideal spanned by the initial polynomials p_i. The XL algorithm is presented in [44] as Algorithm 3.2.

Example 3.2. *Suppose we want to solve the following system of polynomial equations in* $\mathbb{F}_7[x, y, z]$.

Algorithm 3.2 XL

1: **Multiply**: Generate all the products $\prod_{j=1}^{k} x_{i_j} \cdot p_i \in I_D$ with $k \leq D-2$.
2: **Linearize**: Consider each monomial in the x_i of degree $\leq D$ as a new variable and perform Gaussian elimination on the equations obtained in 1.
The ordering on the monomials must be such that all the terms containing one variable (say x_1) are eliminated last.
3: **Solve**: Assume that step 2 yields at least one univariate equation in the power of x_1. Solve this equation over the finite field (e.g., with Berlekamp's algorithm)
4: **Repeat**: Simplify the equations and repeat the process to find the values of the other variables.

$$
\begin{aligned}
x^2 + 5xy + 2xz + 5x + 4y^2 + 6yz + 3y + z^2 + 5z &= 1 \\
3x^2 + 4xy + 6xz + 6x + 2y^2 + 2yz + 4y + 3z^2 + 3z &= 2 \\
2x^2 + 5xy + 4x + 3y^2 + 2yz + y + 2z^2 + z &= 4 \\
6x^2 + 5xz + 3y^2 + yz + 5y + 3z^2 + 2z &= 2
\end{aligned}
$$

Step *1: Let $D = 3$. Then after multiplying all polynomials by all variables, we obtain:*

$$
\begin{aligned}
x^2 + 5xy + 2xz + 5x + 4y^2 + 6yz + 3y + z^2 + 5z &= 1 \\
3x^2 + 4xy + 6xz + 6x + 2y^2 + 2yz + 4y + 3z^2 + 3z &= 2 \\
2x^2 + 5xy + 4x + 3y^2 + 2yz + y + 2z^2 + z &= 4 \\
6x^2 + 5xz + 3y^2 + yz + 5y + 3z^2 + 2z &= 2 \\
x^3 + 5x^2y + 2x^2z + 5x^2 + 4xy^2 + 6xyz + 3xy + xz^2 + 5xz + x &= 0 \\
x^2y + 5xy^2 + 2xyz + 5xy + 4y^3 + 6y^2z + 3y^2 + yz^2 + 5yz + y &= 0 \\
x^2z + 5xyz + 2xz^2 + 5xz + 4y^2z + 6yz^2 + 3yz + z^3 + 5z^2 + z &= 0 \\
3x^3 + 4x^2y + 6x^2z + 6x^2 + 2xy^2 + 2xyz + 4xy + 3xz^2 + 3xz + 2x &= 0 \\
3x^2y + 4xy^2 + 6xyz + 6xy + 2y^3 + 2y^2z + 4y^2 + 3yz^2 + 3yz + 2y &= 0 \\
3x^2z + 4xyz + 6xz^2 + 6xz + 2y^2z + 2yz^2 + 4yz + 3z^3 + 3z^2 + 2z &= 0 \\
2x^3 + 5x^2y + 4x^2 + 3xy^2 + 2xyz + xy + 2xz^2 + xz + 4x &= 0 \\
2x^2y + 5xy^2 + 4xy + 3y^3 + 2y^2z + y^2 + 2yz^2 + yz + 4y &= 0 \\
2x^2z + 5xyz + 4xz + 3y^2z + 2yz^2 + yz + 2z^3 + z^2 + 4z &= 0 \\
6x^3 + 5x^2z + 3xy^2 + xyz + 5xy + 3xz^2 + 2xz + 2x &= 0 \\
6x^2y + 5xyz + 3y^3 + y^2z + 5y^2 + 3yz^2 + 2yz + 2y &= 0 \\
6x^2z + 5xz^2 + 3y^2z + yz^2 + 5yz + 3z^3 + 2z^2 + 2z &= 0
\end{aligned}
$$

Step *2: By linearizing the extended system, we obtain:*

x^2y	x^2z	xy^2	xyz	xz^2	y^2z	yz^2	xy	xz	yz	x^3	x^2	x	y^3	y^2	y	z^3	z^2	z	c
0	0	0	0	0	0	0	5	2	6	0	1	5	0	4	3	0	1	5	1
0	0	0	0	0	0	0	4	6	2	0	3	6	0	2	4	0	3	3	2
0	0	0	0	0	0	0	5	0	2	0	2	4	0	3	1	0	2	1	4
0	0	0	0	0	0	0	0	5	1	0	6	0	0	3	5	0	3	2	2
5	2	4	6	1	0	0	3	5	0	1	5	1	0	0	0	0	0	0	0
1	0	5	2	0	6	1	5	0	5	0	0	0	4	3	1	0	0	0	0
0	1	0	5	2	4	6	0	5	3	0	0	0	0	0	0	1	5	1	0
4	6	2	2	3	0	0	4	3	0	3	6	2	0	0	0	0	0	0	0
3	0	4	6	0	2	3	6	0	3	0	0	0	2	4	2	0	0	0	0
0	3	0	4	6	2	2	0	6	4	0	0	0	0	0	0	3	3	2	0
5	0	3	2	2	0	0	1	1	0	2	4	4	0	0	0	0	0	0	0
2	0	5	0	0	2	2	4	0	1	0	0	0	3	1	4	0	0	0	0
0	2	0	5	0	3	2	0	4	1	0	0	0	0	0	0	2	1	4	0
0	5	3	1	3	0	0	5	2	0	6	0	2	0	0	0	0	0	0	0
6	0	0	5	0	1	3	0	0	2	0	0	0	3	5	2	0	0	0	0
0	6	0	0	5	3	1	0	0	5	0	0	0	0	0	0	3	2	2	0

Step *3: After Gaussian elimination:*

x^2y	x^2z	xy^2	xyz	xz^2	y^2z	yz^2	xy	xz	yz	x^3	x^2	x	y^3	y^2	y	z^3	z^2	z	c
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	6	2	2
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	4	6
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	5	2	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	3	0	5	2	2
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	4	2	5
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	2	0	2	6	2
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	6	0	0	2	2
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	2	5	3
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	2	0	0	2	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	3	0	0	6	2
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	6	4	4
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	6	0	4	4	1
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	2	0	3	2	6
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	4	0	0	5	6
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4	0	0	6	5
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**6**	**3**	**4**

Step *4: Solving* $z^3 + 6z^2 + 3z = 4$ *yields to* $z = 5$.

Step *5: Substituting and solving again we obtain the solution* $x = 1, y = 3, z = 5$.

In the original description of XL, the authors explained how Relinearization can be simplified and lead to XL. They also formally proved that the set of equations defined by a successful Relinearization of degree D is equivalent to a subset of equations derived from the XL algorithm with the same D. Moreover, the matrix size in XL is lower than the matrix size in Relinearization. Thus, XL is able to solve systems that are applicable to Relinearization and more. From mathematical view, Relinearization is about the Ring $K[p_1, \ldots, p_m] \subset K[x_1, \ldots, x_n]$ and XL is about the Ideal $< p_1, \ldots, p_m > \subset K[x_1, \ldots, x_n]$ [122].

Few remarks can be taken into account from the original XL Algorithm 3.2. First, in step 1, there exist a condition "with $k \leq D - 2$". As well as in the notations part, the authors stated "we consider all the polynomials $\prod_{j=1}^{k} x_{i_j} \cdot p_i$ with total degree $\leq D$ ". This means that only quadratic equations are considered. In order to apply XL to a non-quadratic equations, we can simply modify this condition to be " $k \leq D - deg(p_i)$" [8].

Second, in step 2, the authors noted that "The ordering on the monomials must be such that all the terms containing one variable (say x_1) are eliminated last". Thus, there is no explicit known

monomial order. In [8], it is proved that if the XL terminates, it will be also terminated with a lexicographical ordering.

Third, for systems that have multiple solutions or no solutions over their given finite field, XL has no graceful exit behavior. Diem [52] suggested to allow those cases to be ended in failure at the solving step rather than assuming success.

Fourth, the authors failed to demonstrate a termination condition for XL. In [183, 184], Yang et al. discussed when XL can be expected to terminate using combinatorial technique.

Fifth, the complexity of XL is based on the parameter D. So, the authors in [44] and [45] used the heuristic estimation $D \gtrsim \frac{n}{\sqrt{m}}$, for a system with m equations in n variables. They suggested to choose $D \approx \sqrt{n}$ in the case $m \approx n$. If $m = n$ then $D = 2^n$. However, in the case that $m = n + 1$ then $D = n$. For $m = n + c, c \geq 2$, the authors expected to have $D \approx \sqrt{n}$. For systems that have $m = \epsilon n^2, \epsilon > 0$, they expected XL to succeed when $D \approx \lceil \frac{1}{\sqrt{\epsilon}} \rceil$.

All these estimations are based on the assumption that most of the produced equations in XL are linearly independent which looks to work well for special cases, but for general systems this approximation turned out not to be true. As D becomes large, only $\frac{1}{m}$ of the generated polynomials are linearly independent [122].

From the theory of Hilbert-Serre, Moh [122] deduced that the XL algorithm probably works for many interesting cases for D large enough, while XL fails in some cases. Furthermore, Moh stated that the XL method is expected to succeed only if the difference between the number of independent equations of degree $\leq D$ which are generated by step 1 in Algorithm 3.2 and the number of monomials becomes smaller than D. XL was tested against various versions of TTM scheme to find their securities. As a result of these tests, Moh concluded that XL is not effective for many instances of TTM.

As an outcome of analyzing XL and its variants, the authors in [183] derived asymptotic formulas for estimating D, the minimal degree requirements, and for a reliable and successful operations of XL and its variants. Under the assumption of a conjecture of commutative algebra, Diem [52] derived a non-trivial upper bounds on the dimensions of the spaces of equations in the XL algorithm conjectured by Moh [122] and stated in [183]. These upper bounds provide strong evidence that for any fixed finite field K and for any fixed $c \in \mathbb{N}$, the median of the running times of the original XL algorithm applied to systems of $m = n + c$ quadratic equations in n variables over K is not subexponentail in n.

Yang and Chen [182] provided an analysis of the applicability and performance of all XL family for generic systems of equations over medium-sized finite fields. In [143], the authors predicted a formula for the smallest degree for which the XL algorithm will work over $\mathbb{F}_2$. This formula is easier to use than that of Yang and Chen and is restricted to systems which only contain trivial dependencies.

It seems that the bound D is not easily to be theoretical determined without restricted conditions. However, to implement XL under the assumption that we can not know the degree at which a polynomial system could be solved, it is better to start with a smaller value of D and increase it at each failed step. This is so-called incremental XL algorithm. In [8], four ways to realize the process of determining the optimal value of D are suggested. We represent them here starting with $D = \{\max \deg(p_i)\}$ rather than $D = 1$ and P is a system of equations $(p_i = 0)$ to be solved. Then each way is described as follows:

1. Do XL described as in Algorithm 3.2 for P. If you can not obtain the solution, set $D := D + 1$

and do XL again for P with new D.

2. Iterate '**Multiply**' and '**Linearize**' described as in Algorithm 3.2 for P by adding new equations obtained by '**Linearize**' to P. If you can not solve the resulting system, then return to the original P, set $D := D + 1$ and iterate the same procedure as for $D = \{\max \deg(p_i)\}$. Repeat until you obtain the solution.

3. Do XL described as in Algorithm 3.2 for P. If you can not obtain the solution, then set $D := D + 1$, replace P by the resulting system obtained by '**Linearize**' in the previous XL and do XL again for the new P and D. Repeat until you obtain the solution.

4. Iterate '**Multiply**' and '**Linearize**' described as in Algorithm 3.2 for P by adding new equations obtained by '**Linearize**' to P. If you can not solve the resulting system $\bar{P}$, then replace P by $\bar{P}$, set $D := D + 1$ and iterate the same procedure as for $D = \{\max \deg(p_i)\}$. Repeat until you obtain the solution.

A formal description of the fourth one is presented in Algorithm 3.3. This algorithm is applicable to quadratic equations using a Graded lexicographic ordering (grlex) for monomials.

Algorithm 3.3 Formal XL

Inputs
  P: Set of quadratic polynomials with a unique solution.
Output
  $Solution$: The solution of $P = 0$.
Variables
  $\mathcal{M}^{acaulay}$: a matrix whose entries are the coefficients of a system of multivariate polynomial equations in graded lex order.
  $\tilde{P}$: set of all polynomials that are included in the system.
  D: the current degree of $\tilde{P}$.
  $solved$: a flag to indicate whether the system is solved or not.
Begin
Initialization()
  $\{\mathcal{M}^{acaulay}, Solution \leftarrow \varnothing, \tilde{P} \leftarrow P, solved \leftarrow False, solved \leftarrow False, D \leftarrow 2\ \}$
repeat
  $\mathcal{M}^{acaulay} \leftarrow$ **Linearize**($\tilde{P}$)
  ($solved$,$Solution$) $\leftarrow$ **Solve**($\mathcal{M}^{acaulay}$)
  if $solved$ **then**
    Return ($Solution$)
  end if
  $D \leftarrow D + 1$
  $\tilde{P} \leftarrow \tilde{P} \cup$ **Multiply**(P, D)
until ($solved$)
End

3.4 XL Variants

The XL algorithm has had a lot of interest in recent years since it was applied to try finding the secret key of AES. There are several variants of this algorithm. In this section, we briefly describe each variant.

FXL

In [44] an improvement of XL called FXL is proposed. The "F" in FXL stands for "fix". The main idea in FXL is to guess the values of a few variables in the hope that the degree needed for XL will be decreased then applying XL. In FXL, in order to guess the values of r variables, we multiply by 2^r the complexity of XL applied to a system of m equations with $n-r$ variables. The existence of an optimal number of variables to be guessed for the FXL when $m = n$ is discussed in [182].

XFL

The XFL variant was initially named "improved FXL" in [44], while in [182] named as XFL. It means do extend then fix. XFL is applied by the following steps:

1. The n variables are divided into f ("to fix") variables and $n - f$ ones. Multiply the equations by all monomials up to degree $D - 2$ in the $n - f$ non ("to fix") variables only.

2. Order the monomials so that all monomials of exactly degree D with no ("to fix") factor comes first. Eliminate all such monomials from the top-degree block.

3. Substitute actual values for ("to fix") variables, then collate the terms and try to continue XL, re-ordering the monomials if needed, until we find at least one solution.

XL'

The XL' algorithm was proposed in [45]. XL' operates just as XL, but in the last step, instead of requiring at least one equation in only one variable, the hope is to obtain at least r equations with only r variables. Such a system will have on average one solution, and will be solved by the brute-force.

XL2

XL2 was also proposed in [45] as an alternative to XL over $\mathbb{F}_2$. The idea of XL2 is to obtain at least one equation with a restricted set of monomials. From this equation, we may obtain a new equation that was not in the original system. By combining the new equations with the old equations, we obtain another new equations of a special form. This will be repeated as many time as possible. The new extra equations is obtained via the so-called T' method.

Let the set $\tau = \tau^{(D)}$ comprises all monomials of total degree $\leq D$ and T be their number. Let x_1 be one variable. Suppose that τ' is the set of all monomials t_i that are in τ such that we also have $x_1 \cdot t_i \in \tau$ and T' be their number, i.e. $T' =\mid \tau'_i \mid$, where $\tau'_i = \{\mathbf{x}^b : x_i\mathbf{x}^b \in \tau\}$ for each i.

Assume that we have $Free \geq T - T' + C$ with $C > 1$. Then XL2 applies the following steps in order to generate more useful equations [183]:

1. Starting from the equations $\mathcal{R} = \mathcal{R}^{(D)}$, eliminate monomials not in τ_1' first. We are then left with relations $\mathcal{R}_1$, which gives each monomial in $\tau \setminus \tau_1'$ as a linear combination of monomials in τ_1, plus C equations $\mathcal{R}_1'$ with terms only in τ_1'.

2. Repeat for τ_2' to get the equations $\mathcal{R}_2$ and $\mathcal{R}_2'$ (we should also have $\mid \mathcal{R}_2' \mid = C$).

3. For each $\ell \in \mathcal{R}_1'$, use $\mathcal{R}_2$ to write every monomial in $\tau \setminus \tau_2'$ in the equation $x_1\ell = 0$ ($\ell \in \mathcal{R}_2'$) in terms of those in τ_2. Do the converse for each $x_2'\ell$. Then we get $2C$ new equations.

XLF

XLF was proposed in [42]. It is designed for the case that $m \approx n$ and over the finite field $\mathbb{F}_{2^k}$. XL stands for multiply (X) and linearize (L), while XLF, stands for multiply (X) and linearize (L) and apply Frobenius mappings (F).

The idea is to try to use the Frobenius relations $x^q = x$ to take advantage when $q = 2^k$, by considering (x_i), (x_i^2), (x_i^4), $\ldots$, $(x_i^{2^k-1})$ as new independent variables, replicating all $\sum_{ij} \alpha_{ij} x_i x_j = 0$ equations k times by repeatably squaring them, i.e. $\sum_{ij} \alpha_{ij}^2 x_i^2 x_j^2 = 0$, then trying to use the equivalence of the identical monomials as extra equations.

XSL

The XSL, stands for eXtended Sparse Linearization, method was introduced in 2002 by Courtios and Pieprzyk. The preeminent idea of XSL is to manipulate the structure of some types of block ciphers, for example the AES and Serpent. These ciphers can be described by sparse as well as overdetermined systems of multivariate polynomial equations. On the other hand, XL could not make a good use of such sparse structure of the system, since XL has no good strategy of choosing monomials to be multiplied.

The difference between XL and XSL is that the XL algorithm will multiply a system of equations by every possible monomial of degree at most $D - 2$, where D is fixed. The XSL algorithm suggests multiplying the system of equations only by carefully selected monomials, for example equations are only multiplied by monomials that are already appear in other equations. Thus, XSL can create fewer new monomials when generating the new equations. Additionally, there is a last step (called T' method), in which the algorithm try to obtain new linearly independent equations without creating any new monomials.

There are three different versions of XSL. The first two, known as "eprint version", were presented in [47] and the third one, sometimes referred to as "compact XSL", was introduced in [46]. According to the authors claims, the first XSL attack of the eprint version was very general, it did not use the cipher's key schedule but more plaintext-ciphertext pairs are required, and was studied approximatively in order to investigate the asymptotic behavior of XSL. The second XSL attack used the key schedule with less plaintext-ciphertext pairs and was designed for concrete cryptanalysis of AES and Serpent. The compact XSL was supposed to work only on special types of ciphers.

As an early criticism to XSL, the authors in [130] believed that the XSL estimates did not have the accuracy needed to substantiate claims of the existence of an AES key recovery attack based on XSL algorithm. The effectiveness of XSL attacks and their applicability to block ciphers had been discussed in [181]. The condition of XSL for a block cipher was discussed in [110] with a conclusion that the XSL is feasible if each system of linearized equations for every s-box is overdetermined. In [113], an analysis of the second XSL attack on BES over $\mathbb{F}_{256}$ was presented. The main conclusion for that analysis was that if XSL works on BES, then it is worse than brute force. The same conclusion in [31] was stated that the compact XSL attack is not an effective attack against AES cipher. On the contrary, Qu and Liu [137] proposed that the compact XSL attack on BES-128 with the key schedule involved is applicable and gives results better than that of [113].

Furthermore, Ji and Hu in [96] extended the SMS4 cipher to a new cipher ESMS4 over $\mathbb{F}_{256}$ by conjugate transformation. By using this transformation, Ji and Hu were able to describe the SMS4 cipher with an extremely sparse overdetermined multivariate quadratic system over $\mathbb{F}_{256}$. Thus, it is possible to use XSL algorithm to estimate the security level of the SMS4 cipher. As a result, they found that the complexity of solving the whole system with the eprint version of XSL algorithm is 2^{77}. The analysis on the eprint version on BES in [113] and ESMS4 in [96] was adapted to the compact version in [30]. These adaptations led to the following result: the compact XSL attack on both BES and ESMS4 are worse than exhaustive search.

AES is a very important block cipher. It is already globally used in commerce and government for the transmission of secrete information. Breaking such a cipher is a nightmare. There exist a great controversy between cryptanalysts on whether the XSL attack works or not. The main conclusion in this context is that XSL is not a valid technique. We end this section with Vincent Rijmen's, one of the AES co-creators, proverb "The XSL attack is not an attack. It is a dream" as well as Courtois's answer "It will become your nightmare".

Linear Method

Another variant, called Linear Method, was described in [98]. The basic concept of this method is that like XL tries to find univariate polynomials, the Linear Method looks for linear polynomials in the ideal. As soon as sufficiently many linearly independent linear polynomials are found, the solution of the system could be found. The linear method was used to break the TRMC scheme, a multivariate-based public key cryptosystem [170].

The HXL Algorithm

The HXL algorithm was presented in [88]. The H of the HXL algorithm stands for "Heuristic" and "Hybrid, with F4". The main idea is to define a variant of XL that is capable to compute a Gröbner basis of a set of a multivariate quadratic polynomial equations over $\mathbb{F}_2$ without using S-polynomial as in F4.

The ElimLin Algorithm

The ElimLin algorithm was proposed by N. Courtois [38]. It was used to attack DES (breaks 5-round DES). The main idea is to obtain all the linear equations in the span of initial equations

by using Gaussian elimination, then one of the variables nominated in each linear equation and is substituted in the whole system. These processes are repeated up to the time no new linear equation is found.

3.5 XL versus other solvers

Currently there are many different algorithms which are based on different strategies to solve systems of multivariate quadratic polynomial equations. Gröbner basis methods and linearization are the two common categories to solve such systems. Semaev and Mikuš [152] present another way to classify these algorithms which depends on the equation representation. This classification falls into three categories: Gröbner basis algorithms, SAT-Solving methods and Agreeing-Gluing algorithms.

In this section we briefly present the most known methods for solving systems of multivariate polynomial equations over finite fields that are used in algebraic cryptanalysis. The relation and/or comparison between these methods and XL is also discussed.

The Gröbner basis Algorithms

The general strategy of the Gröbner basis algorithms is to transfer a generating set for an ideal to another generating set of polynomials, called Gröbner basis, with certain nice properties [21]. The first algorithm and the notation for Gröbner basis was introduced by Buchberger in [20]. It is based on adding the normal form of the so-called S-polynomials to the set of original polynomials that we aim to solve. The most important two improvements of the Buchbereger's algorithm are the F4 [72] and F5 [73] algorithms by Faugère. A matrix version of F5 called MatrixF5 was mentioned in [14] but not specifically described. MatrixF5 was presented in several French PhD theses, an English description of this algorithm can be found in Martin Albrecht PhD thesis [2]. F5/2 [75] is a modified algorithm of F5 for calculating Gröbner basis over $\mathbb{F}_2$ in the case where the original system contains the field equations. Recently, a variant of F4 algorithm was presented in [99] which is an adaptation of the "Gröbner trace" method of Traverso [167] to the context of F4. In [68], Eder and Perry introduced a variant of F5 called F5C which replaces each intermediate Gröbner basis with its reduced Gröbner basis. The G^2V was presented in [81] as a simpler and more efficient algorithm than F5 and F5C. The extension of F5 to any input set of polynomials, not only homogeneous or regular, was discussed in [188]. Recently in [5], Albrecht and Perry presented the F4/5 algorithm which combines F4-style reduction with the F5 criteria.

The comparison between XL and Gröbner basis algorithms as well as the relation between them was studied in [8, 74, 162]. The authors of [8] showed that to solve a system of algebraic equations treated in XL is equivalent to calculate the reduced Gröbner basis of the ideal associated to the system. Taking into account that, the system we are speaking about has a unique solution and the finite field equations are applied in the $\mathbf{Multiply}$ process. Furthermore, they gave an F4-like detailed description of the XL algorithm. In this description XL is represented as a redundant variant of F4 algorithm. Indeed, in this context, XL is merely a slower implementation of Gröbner basis algorithms [60]. A more detailed version with proofs of [8] is represented in [74] while another version with an additional idea that the T' method in XSL algorithm can be interpreted in terms of Buchberger's algorithm is represented in [162].

The GeometricXL Algorithm

The GeometricXL algorithm [128] is an algorithm which generalizes the XL algorithm to a geometrically invariant algorithm. GeometricXL can be considerably more efficient in some cases than either XL or Gröbner basis algorithms. These cases involve homogeneous polynomial equation systems that are solvable at some degree D to be less than the characteristic of the finite field. Although the ability of GeometricXL to solve certain systems at lower degree than XL or Gröbner basis, it is not well-suited to fields of even characteristic. An adaption of the GeometricXL algorithm to even characteristic, which termed as EGHAM process, is described in [129].

The DR Algorithm

Based on extended Dixon Resultants, an algorithm called DR [163] was proposed to solve systems of multivariate polynomial equations by taking $x_1, \ldots, x_{n-1}$ as variables and x_n as a parameter. By comparing the run time of the DR algorithm with the Buchberger's algorithm in Maple 9.5, the inventors of DR claimed that their algorithm is far more efficient than Buchberger. The matrix size between DR and FXL was also compared. For small instances that were used, DR is shown to have a smaller matrix size. The proof for the correctness and termination of the DR algorithm was stated in [77]. A modified version of DR that terminates for all equation systems was presented in [1].

The Raddum-Semaev Algorithms

Raddum and Semaev in [139, 140] presented different approaches for solving sparse non-linear equation systems. These approaches depend on representing the equation systems in a different way and focusing on the solution set. In [139], the equations are represented as lists of bit-strings, where each string is a value assignment of variables that satisfies the equations. The solution can be seen as message-passing on a graph. In [141], the equations are represented as a system of Multiple Right Hand Sides (MRHS) linear equations. The solution can be found using a generalized Agreeing and Gluing techniques from [148]. A more algebraic and geometric interpretation of [139] can be found in Diplomarbeit [67] and discussions of improving the Agreeing-Gluing algorithm existed in [149, 151]. Furthermore, a description of the Agreeing algorithm as a hardware implementation with a lattice of circuits is presented in [150]. To summarize, the key idea for this approach is to look at another way of representing non-linear equations. With this new representation, new ways for solving systems of equations are followed.

The Zhuang-Zi Algorithm

Ding, Gower, and Schmidt presented the Zhuang-Zi algorithm [58], a method for solving multivariate polynomial equations over a finite field. The main idea of this algorithm is to convert the problem of solving a set of multivariate polynomial equations into a univariate equation over a large extension field using the Frobenius map. The authors described the algorithm and presented some interesting examples, where this algorithm works and the other known fast algorithms do not.

The SAT-solvers

A new trend in solving systems of Boolean multivariate polynomial equations over is to use SAT-solvers. A software program, which given a logical sentence written in the propositional logic, tries to find an assignment of True and False to each variable to make the entire sentence come out True is known as a SAT-solver. Recently, with the rapid growth of the computational power, SAT-solvers have been enabled to deal with problems in a much larger scale.

In Boolean logic, Boolean variables are variables whose value either True or False; a variable or a negation of a variable is referred as a literal. Several Boolean variables can be combined by using a conjunction (logical AND), a disjunction (logical OR), and a complement (logical NOT) as a Boolean formula. Such a formula that contains only OR and NOT operators is called a clause. A formula that contains a combination of clauses using AND operator is known as CNF (Conjunction Normal Form).

Given a Boolean formula, the problem of determining whether there exists an assignment of values to the variables such that the formula holds is called the satisfiability problem (or SAT-problem). This problem was proved to be NP-complete problem [33].

Every instance of a problem in NP class can be transformed into a SAT-instance in polynomial time, since all NP-complete problems are polynomially equivalent. Inspired by this equivalence, either MQ-problem or SAT-problem can be an efficient tool for the solution of each other, cryptanalysts investigate to employ SAT-solvers in the field of cryptanalysis. In the other hand, a cryptographic algorithm can be modeled as a SAT-problem.

In algebraic cryptanalysis, the main task for the cryptanalyst is to convert the equation system in the Algebraic Normal Form (ANF) to the Conjunctive Normal Form (CNF). Then to solve the SAT-problem using a SAT-solver. There are four research aspects that use SAT-solvers [28]. The first one spotlights on the preprocessing of the equation system. The second one concerns with converting of the equations from ANF to CNF, different approaches can be found in [101, 131]. The preprocessing of CNF is considered as the third one. The last one discusses the improving of the SAT-solvers performance using different guessing strategies.

The first connection between SAT and Cryptography dates back to [34], where a suggestion appeared to encode public-key algorithms such as RSA into propositional logic. The first application of SAT-solvers in cryptanalysis was due to Massacci et al. who successfully attacked three rounds of DES after modeling it as a SAT-problem [115, 116] and also explored forging an RSA signature [78]. While in [38], 6 rounds of DES were attacked using a more algebraic approach. Courtois et al. presented an attack on the Keeloq block cipher [39]. In [71], the block cipher SMS4 as well as a simplified version of it was analyzed using SAT-solvers. SAT-solvers have also been applied to attack stream ciphers like Hitage2 [40], MiFare Crypto1 [80], and a modified version of Trivium called Bivium [70].

The application of SAT-solvers to the cryptanalysis of hash functions started in [100], where a suggestion stated to convert the hash function objectives into a logical expressions. In [119] , Mironov and Zhang were able to generate a full collisions for MD4 and MD5 hash functions using SAT-solvers. The MD4 have been subjected to inversion attacks using SAT-solvers in [49]. CubHash and KECCAK, two candidates for the NIST SHA-3 competition, were analyzed in [144] and [127], respectively.

All the applications of SAT-solvers to the field of cryptanalysis try to customize the problem description to be suitable to the solver language. Therefore, the cryptanalysts use the SAT-solvers

as black boxes. Applying the opposite direction, i.e. customizing the solver to match the cryptographic primitive, is a candle on the road. In [160], Mate Soos et al. presented several steps towards a specialized SAT-solver for cryptography called CryptoMiniSat which is an enhanced version of the Minisat solver [69].

The MIP-based Algorithm

The minimization or maximization of a linear function in several variables, subject to linear equality and inequality constraints as well as integral restrictions on some or all the variables, is a well-known problem in the field of combinatorial and integer optimization. This problem is known as the Mixed-Integer Programing (MIP) problem or MILP-problem (Mixed-Integer Linear Programing) and it is believed to be generally NP-hard [86]. The feasibility or optimality of an MIP-problem instance can be decided using off-the-shelf MIP solvers, a list of such solvers can be found in [114].

Transforming the problem of solving a sparse system of quadratic equations over $\mathbb{F}_2$ into an MIP-problem was discussed in [111]. In [16], Julia Borghoff et al. presented Bivium A and Bivium B [138] as an MIP-problem by using two different methods for converting Bivium into an MIP-problem. The Integer Adapted Standard conversion method was discussed and implemented in [2];

The CS-based Algorithms

The Characteristic Set (CS) method, or Wu's method, is an approach to analyze and solve polynomial equation systems. According to [83], the main idea of this method is to reduce an equation system over an algebraically closed field in general form to equation systems in a special "triangular form", also called ascending chains. The zero-set of any finitely generated equations can be decomposed into the union of the zero-sets of ascending chains. As a consequence, solving an equation system can be reduced to solving cascaded univariate equations.

Xiao-shan Gao et al. [83], extended the general CS method to solve nonlinear equation systems over the finite field $\mathbb{F}_2$ with an application to stream ciphers that are based on nonlinear filter generators. The same authors presented an additional as well as an improvement of the CS methods over $\mathbb{F}_2$ described in [27]. An extension of the CS method to solve equations in any finite field $\mathbb{F}_q$ can be found in [84, 85]. In [153], two algorithms called TDCS and MFCS were presented to solve nonlinear equation systems in finite fields based on the idea of characteristic set. The MFCS algorithm was tested to solve boolean equations generated from Bivium A in [138].

3.6 XL Implementation

The XL Algorithm 3.3 has been implemented in C/C++ based on different versions of the M4RI (pronounced "Mary") package [3]. M4RI is a library for fast arithmetic with dense matrices over the finite field $\mathbb{F}_2$. The name M4RI comes from the first implemented algorithm: The "Method of the Four Russians Inversion" algorithm [9] published by Gregory Bard. M4RI is used by the Sage mathematics software [161] and the PolyBoRi [19]. Furthermore, M4RI will be soon an optional package in the Debian distribution of Linux [12].

The reduction of a matrix over the field with two elements ($\mathbb{F}_2$) into Row-Echelon Form (REF), or Reduced-Row-Echelon Form (RREF) is the backbone of solving polynomial systems using

XL. Gaussian elimination is a very well-known and long-standing classical algorithm for this reduction using row operations. The only problem with that algorithm is its complexity $\mathcal{O}(n^3)$, where n is the matrix dimension. This cubic-time complexity makes it far too slow in practical for dense systems. In addition to the naive Gaussian elimination algorithm, the new version of M4RI package, namely 20100817, provides two algorithms for computing echelon forms: M4RI algorithm and PLS decomposition.

The M4RI algorithm was presented in [9] and also was discussed in more details in [10, 12]. M4RI is an aggregation of the Method of Four Russians for matrix Multiplication (M4RM) algorithm and Gaussian elimination with a theoretical complexity of $\mathcal{O}(n^3/\log n)$. The M4RI algorithm is not only used for matrix inversion, but also is used over $\mathbb{F}_2$ for LUP-factorization of a matrix, determining the rank of a matrix, solving a system of linear equations, and reducing a matrix to REF or REEF. The procedure "$mzd\_echelonize\_m4ri$" in the M4RI package (version 20100817) is the implementation to return the (reduced) row echelon form and the rank of a matrix.

Matrix decomposition refers to the transformation of a given matrix into a given canonical form. When the given matrix is transformed to a right-hand-side product of canonical matrices, the process of producing this decomposition is also called "matrix factorization". There are many different matrix decompositions which have several applications. The PLUQ, LQUP, PLS, and LPS are examples of such decompositions [93, 95]. In the M4RI package (version 20100817), the PLS decomposition, which is equivalent to PLUQ as well as has several advantages over $\mathbb{F}_2$ [4], is the second method to compute the REF or RREF. The procedure "$mzd\_echelonize\_pluq$" in the M4RI package (version 20100817) is the implementation to return the (reduced) row echelon form and the rank of a matrix.

A hybrid procedure, called "$mzd\_echelonize$", is also presented in the M4RI package. This procedure starts with the M4RI algorithm and switch over the PLS as soon as the density of the remaining part of the matrix reaches a predetermined percentage, for more details see [4].

The performance of the M4RI package depends on the particular machine configuration and architecture. The best results could be obtained when the M4RI package runs on 64-bit x86 architectures (x86-64), specifically the Intel Core 2 and the AMD Opteron [2]. This assumption does not prevent the package to also run on 32-bit CPUs and on non-X86 CPUs such as the PowerPC. Therefore, the performance of the M4RI package directly affects the performance of the XL algorithm.

The second important process in XL is to extend the system up to some degree D by multiplying the original polynomials by monomials of degree $D-2$ for the case of quadratic polynomials. In the matrix form of representing a polynomial, each monomial has a unique column index. In order to extend a polynomial up to D, we apply three steps to obtain the new extend polynomial. The first step is to convert each column index to another indexed family of integers which represent the index of each variable in the monomial. For example, "1", the index of the monomial x_1x_2 in a graded lexicographic ordering, is converted to $< 1, 2 >$. The second step is to combine each indexed family of integers with all the monomials of degree $D-2$ taking into account the field equations effects. For example, to multiply x_1x_2 by x_2x_3, we combine $< 1, 2 >$ with $< 2, 3 >$ to obtain $< 1, 2, 3 >$ which corresponds to $x_1x_2x_3$. The last step is to convert the new indexed family of integers to its corresponding column index in the extended polynomial with degree D, $< 1, 2, 3 >$ will be converted to index "2" at degree 3. The bijective mapping between the indexed family of integers and the column index is implemented directly using the Rankcomp and

Unrankcomp algorithms in [109].

3.7 XL Experimental Results

In this section we present experimental results of the XL algorithm and compare the different algorithms that are implemented in the M4RI package. In particular, we compare the M4RI algorithm and the PLS decomposition based echelon form in terms of memory usage and time. These experiments are based on the latest version of M4RI package which is released on August 2010. On the other hand, we compare the XL results with the Magma's implementation of the F4 algorithm. It is known that Magma [17, 26] is the best computational algebra system in implementing F4 [72]. The F4 results are based on the latest version namely V2.16-8 released on April 2010.

We are interested in solving systems of multivariate quadratic polynomial equations when the number of equations is the same as the number of unknowns over $\mathbb{F}_2$. We use some instances of dense random systems generated by Courtois [43]. Specifically, the instances from number of variables equal to 10 to number of variables equal to 24. The instances from 25 variables to 31 variables were generated according to the description in [45, Section 5]. All of these instance have a unique solution.

We give an average running time over ten trails for all instances except for the instances with 27, 28 and 29 variables, the average of three is used since these instances need more time. Moreover, the complexity of solving systems of dense multivariate quadratic polynomial equations depends on the number of variables and the number of equations in each system. Therefore, the complexity of solving different systems with the same number of variables and equations more or less will be the same.

All the experiments are done on a Sun X4440 server, with four "Quad-Core AMD Opteron^{TM} Processor 8356" CPUs and 128GB of main memory. Each CPU is running at 2.3 GHz. In these experiments we use only one out of the 16 cores. This server is the so-called Userver2 which belongs to the Cryptography and Computer Algebra group, Computer Science Department, TU-Darmstadt.

In Table 3.1 and Table 3.2, the first column "Sys" denotes the system type, MQR as abbreviation for Multivariate Quadratic Random, followed by the number of variables, which is the same as the number of equations, of the initial system. The maximum degree bound that the system can reach is denoted by "D". The number of rows and columns of the largest linear system to which Gaussian elimination is applied are represented by "nrows" and "ncols", respectively. The rank of this matrix can be found in the "Rank" column. The total number of the nonzero elements in the system up to degree D without applying Gaussian at all is denoted by "NonZero". For each algorithm (M4RI, PLS), the used memory in Megabytes (MiB), Gigabytes (GB) and the execution time in Seconds (S), Minuets (M), Hours (H), and Days (D) are represented by "Mem" and "Time", respectively.

Table 3.1 shows that the largest system that could be solved by XL is a dense random system of 29 multivariate quadratic polynomial equations in 29 variables using 128GB RAM. Table 3.2 shows that the largest system is a system of 31 equations in 31 variables for the F4 algorithm. Figure 3.1 shows the nonzero elements in the matrix generated for the MQR13 instance at degree 4 without performing any Gaussian elimination. Figure 3.2 presents the same matrix after applying Gaussian elimination to produce a row echelon form to the matrix at degree 4.

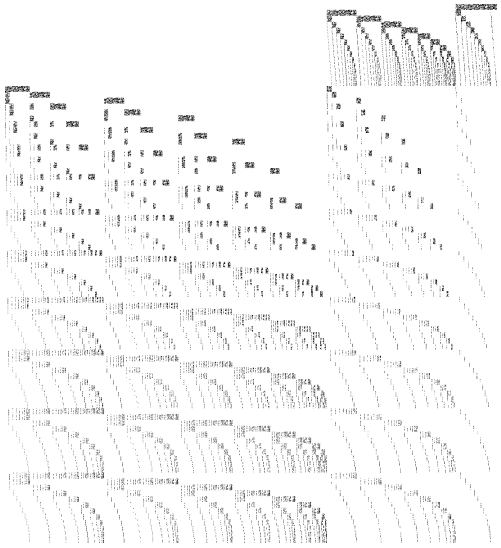

Figure 3.1: MQR13 Matrix before Gaussian elimination

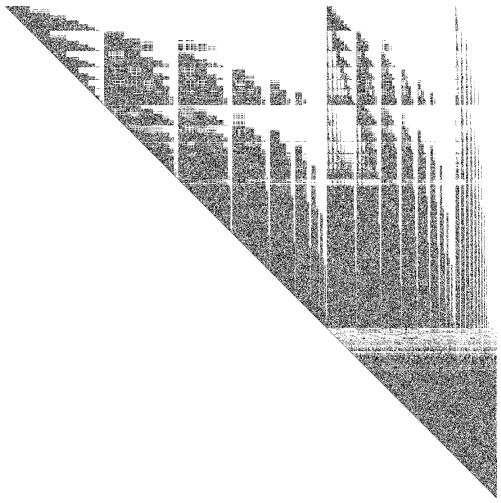

Figure 3.2: MQR13 Matrix after Gaussian elimination

Table 3.1: Performance of XL for Dense Random Systems

Sys	D	nrows	ncols	Rank	NonZero	M4RI		PLS	
						Mem	Time	Mem	Time
MQR10	4	560	386	385	10698	3MiB	1S	3MiB	1S
MQR11	4	737	562	561	18296	3MiB	1S	3MiB	1S
MQR12	4	948	794	793	27381	4MiB	1S	4MiB	1S
MQR13	4	1196	1093	1092	40834	4MiB	1S	5MiB	1S
MQR14	5	6475	3473	3472	231824	11MiB	1S	11MiB	1S
MQR15	5	8520	4944	4943	343836	18MiB	2S	18MiB	2S
MQR16	5	11016	6885	6884	533244	29MiB	3S	29MiB	3S
MQR17	5	14025	9402	9401	765590	47MiB	5S	47MiB	5S
MQR18	5	17613	12616	12615	1117752	77MiB	9S	77MiB	7S
MQR19	5	21850	16664	16663	1528734	124MiB	16S	136MiB	13S
MQR20	5	26810	21700	21699	2138779	198MiB	29S	217MiB	22S
MQR21	6	153405	82160	82159	12535370	3.6GB	36M	3.6GB	22M
MQR22	6	194579	110056	110055	17421636	6.2GB	1.5H	6.2GB	43M
MQR23	6	244145	145499	145498	24345424	10.3GB	3.5H	10.3GB	2.1H
MQR24	6	303324	190051	190050	33610950	14.5GB	10.9H	17.0GB	2.8H
MQR25	6	373450	245506	245505	44795794	20.3GB	18.7H	27.3GB	5.9H
MQR26	6	455975	313912	313911	59438768	43.1GB	1.5D	42.8GB	10.5H
MQR27	6	552474	397594	397593	78495884	53.5GB	3.1D	67.0GB	16.9H
MQR28	6	664650	499178	499177	103480412	85.8GB	5.7D	102.5GB	1.5D
MQR29	6	794339	621616	621615	132637356	$>$ 118GB	$>$ 15D	124.4GB	2.8D

Table 3.2: Performance of F4 for Dense Random Systems

Sys	$F4_{v2.16-8}$				
	D	nrows	ncols	Mem	Time
MQR10	4	624	339	9MiB	1S
MQR11	4	812	511	9MiB	1S
MQR12	4	1010	709	9MiB	1S
MQR13	4	1261	990	9MiB	1S
MQR14	4	1568	1366	10MiB	1S
MQR15	4	2641	1537	11MiB	1S
MQR16	5	10009	4050	21MiB	1S
MQR17	5	12382	5784	32MiB	1S
MQR18	5	15187	8120	47MiB	1S
MQR19	5	18453	11053	71MiB	2S
MQR20	5	22441	14979	106MiB	3S
MQR21	5	26915	19811	158MiB	22S
MQR22	5	63631	21865	356MiB	47S
MQR23	5	38011	33165	503MiB	2M
MQR24	6	207150	78637	3.5GB	4M
MQR25	6	249003	109254	7.4GB	5M
MQR26	6	299897	150109	11.6GB	12M
MQR27	6	356966	200679	17.3GB	24M
MQR28	6	419268	259655	21.6GB	54M
MQR29	6	499424	340456	31.6GB	1.9H
MQR30	6	1284049	374261	92.9GB	15.5H
MQR31	6	865972	487060	113.2GB	1.2D

3.8 Conclusion

In order to improve the XL algorithm we need to understand the basic facts that are associated with this algorithm. In particular, we need to determine the limit that can be achieved by XL. In order to find such a limit, we look how far can we go in solving random dense quadratic systems. In this chapter, we discussed the XL algorithm from several directions and presented the needed background material for this algorithm.

From a practical point of view, we are able to solve dense random systems up to 29 multivariate quadratic polynomial equations in 29 variables on a server that has 128GB memory. On the other hand, the Magma's implementation of the F4 algorithm is able to solve a dense random system with 31 multivariate quadratic polynomial equations in 31 variables on the same server.

Beside the shortage in the limit of XL compared to F4, there exist some systems that could be solved by XL at a degree greater than the degree at which F4 can solve. Dense random systems with number of variables and equations equal to 14, 15, 21, 22, and 23 are examples of such systems that are solved by F4 at a degree smaller than XL. Other examples are most of the systems obtained from multivariate-based cryptosystems, see Chapter 2.

4 Mutant-based Improvements

In this chapter, we discuss the idea of mutant and state Ding's concept of mutant. The impact of this concept in solving systems of multivariate polynomial equations is presented as well as the definition of a mutant. The practical application of the idea of mutants is presented in two algorithms, namely the MutantXL algorithm and the MXL2 algorithm. The experimental results of using both algorithms are introduced. These experiments are for dense random systems and HFE systems over $\mathbb{F}_2$. This is followed by the practical algebraic cryptanalysis of the two multivariate public key schemes Little Dragon Two and Poly-Dragon using MXL2.

4.1 Introduction

The intractability of solving large systems of multivariate quadratic polynomial equations over finite fields is the security basis for multivariate-based public-key cryptosystems (MPKCs). The problem of solving such systems is called the MQ-problem. It was proven that the general MQ-problem is NP-complete, see Chapter 2 for more details. From a cryptanalysis point of view, constructing a system of nonlinear multivariate polynomial equations over finite fields that defines a secret of a cryptographic primitive and then solving this system to recover that secret is called algebraic cryptanalysis (attack). These attacks are not only applicable to MPKCs but also applicable to a variety of ciphers.

Practically, the problem of solving the corresponding multivariate system for some cryptographic schemes has been demonstrated to be easier than random instances of the MQ-problem which allows these schemes to be broken. Therefore, algorithms for solving such systems are important tools for cryptanalysis.

Several algorithms have been proposed to find solution(s) for systems of multivariate polynomial equations over finite fields. These algorithms can be classified into two research directions. The first one is the Gröbner basis algorithms such as the F4 algorithm. The second direction is the linearization-based algorithms such as the XL algorithm.

The concept of Gröbner bases was introduced by Bruno Buchberger as early as 1965. The timeline of Gröbner basis algorithms dates back to the Buchberger's original algorithm. A lot of improvements and research were presented during the decades until we reached to the F4 and F5 algorithms which were presented by Faugère in 1999 and 2002, respectively. These two algorithms are the most important tools in computer algebra and are the best known efficient algorithms for solving systems of polynomial equations. Other algorithms such as F5C [68], G2V [81] and GVW [82] for computing Gröbner basis were proposed after 2002 until now (2011). The common factors of these algorithms are the lack of efficient implementation and the absence of breaking real world challenges using these algorithms.

On the other hand, in cryptography, another algorithm for solving systems of multivariate polynomial equations over finite fields, called the XL algorithm, was introduced by Nicolas Courtois

et al. in 2000. After the presentation of the XL algorithm, some variants of XL were introduced. This was followed by some publications in an attempt to understand the behavior of XL, its complexity, its relation to Gröbner basis algorithms as well as the comparison between them and in what context XL could be used. For more details about XL, see Chapter 3.

From a practical point of view, XL is not as efficient as Gröbner basis algorithms. The running time and memory consumption for XL is greater than in the case for the F4 algorithm. We have to take into account that the research on Gröbner bases has started since four decades while for XL, it was one decade ago. In this context, improving the XL algorithm is a promising research area.

In this chapter, we present an idea of Jintai Ding to improve algorithms that are based on generating elements in an ideal, including the main idea, definitions, and the different applications to use that idea. We focus only on two applications to improve the XL algorithm. The two applications, namely the MutantXL and MXL2 algorithms, are the main contribution of this chapter. This is followed by the practical evidences for the efficiency of each algorithm compared to XL and/or F4 algorithms. The use of MXL2 in the context of algebraic cryptanalysis is also presented by breaking two MPKCs, namely the Little Dragon Two and the Poly-Dragon cryptosystems. The conclusion of this chapter is introduced at the end of the chapter. The discussions, design, implementation and testing for the MutantXL and MXL2 algorithms in this chapter is a joint work with Mohamed Saied Emam Mohamed.

4.2 Ding's Concept of Mutant

There are many different algorithms which are based on different strategies to solve systems of multivariate polynomial equations. Gröbner basis methods and linearization are the two common categories to solve such systems of equations, see Chapter 3 for more details. The basic idea behind these two categories is to start with an original set of multivariate polynomial equations. Then, we need to define a way to produce new polynomials in the ideal generated by the original polynomials, a way to reduce existing polynomials, and a way to check whether a solution has been found or not. One way to improve these algorithms is to generate new low degree polynomials that are in the ideal generated by the original polynomials with the hope that these new polynomials are linearly independent with the existing polynomials.

In this section, we present an idea of Jintai Ding to extract such lower degree polynomials from the linear algebra step. After that, the mathematical definition of such lower degree polynomials is introduced. This is followed by an overview of different applications that use these lower degree polynomials.

Idea of Mutant

The basic idea of the XL algorithm is to extend the linearized system of an initial system of polynomials to find univariate polynomial(s). The extending process is done by multiplying each initial polynomial by all the monomials up to some degree. This process is repeated by increasing the degree one by one until a solution is obtained. Figure 4.1 represent the general processes in the XL algorithm.

The overall performance of XL depends on the degree at which a solution could be found. The lower the degree at which a solution could be found, the better the performance of XL. Indeed, we

want to extend the initial system on one hand as much as possible to obtain a solution and on the other hand as little as possible to obtain a better performance in terms of time and memory.

In this context, Jintai Ding [53] noticed that after applying Gaussian elimination to the extended system at some degree, there exist some new polynomials of smaller degree than expected. At this degree, if there is no univariate polynomial, XL did not pay attention to such new lower degree polynomials. Then, XL started to extend the system to the next higher degree. In other words, XL treats old and new generated polynomials as if they are all of the same degree.

As a result of this intuition, Ding stated the following concept: *In the process of extending a system to find a solution with a smaller degree, we should treat polynomials of different degrees differently*. In particular, the polynomials of lower degree should play more important role than the ones of the higher degree. The lower the degree the higher the value.

This idea grew from Ding's thoughts in attending IMA2006 and IMA2007 workshops in the subject of applied algebraic geometry. He started to write his first version notes in December 2006 in TUD, Technische Universität Darmstadt, Germany. Ding called the new lower degree polynomials in XL **Mutants**. Therefore, the main idea for the Mutant strategy is to maximize the effect of lower degree polynomials occurring during the linear algebra step.

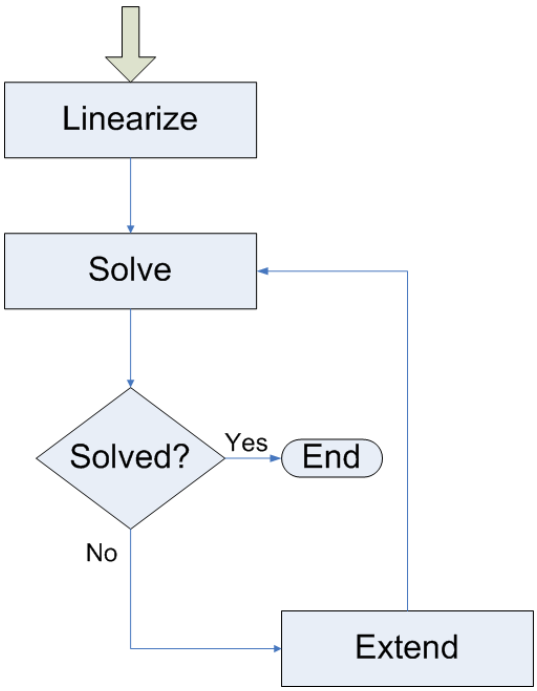

Figure 4.1: General XL

Definition of Mutant

In this section we introduce mutants and explain their importance for solving systems of multivariate polynomial equations. Although almost all the notions and results of this section are true for a polynomial ring over any finite field, we concentrate on the case of the Boolean polynomial

ring, since this case has a special importance for cryptography. Throughout this section we will use $x := \{x_1, \ldots, x_n\}$ to be a set of n variables. Let

$$R := \mathbb{F}_2[x_1, \ldots, x_n]/\langle x_1^2 - x_1, \ldots, x_n^2 - x_n \rangle$$

be the Boolean polynomial ring in x with graded lexicographical ordering $<_{grlex}$ on the monomials of R. We consider elements of R as polynomials over $\mathbb{F}_2$ where the degree of each term w.r.t any variable is 0 or 1. Let $P = \{p_1, \ldots, p_m\}$ be a system of m quadratic polynomials in R. We are interested in finding a solution for the equation system $p_i(x_1, \ldots, x_n) = 0$, $1 \leq i \leq m$ denoted by $P(x) = 0$ that is based on creating further elements of the ideal generated by the polynomials of P. In this context, mutants are defined in Definition 4.1.

Definition 4.1. *Let I be the ideal generated by the finite set of polynomials P, $f \in I$. For any representation of f, $f := \sum_{p \in P} f_p p$, we define the level of this representation to be $max\{deg(f_p p) : p \in P, f_p \neq 0\}$. Let $Rep(f)$ be the set of all representations of f. Then, the $level$ of f with respect to P is defined to be the minimum of the levels of all representations in $Rep(f)$. The polynomial f is called a Mutant with respect to P if its degree is less than its level.*

Note that, if the elements in P are homogeneous polynomial, mutants will never appear. This is due to the fact that multiplying a homogeneous polynomial by a monomial generates a new homogeneous polynomial with higher degree. Therefore, eliminating the leading monomials of a system of homogeneous polynomials leads to new polynomials of the same degree.

When a mutant is written as a linear combination $\sum_{p \in P} f_p p$, then one of the polynomials $f_p p$ has a degree exceeding the degree of the mutant. This means that a mutant of degree d cannot be found as a linear combination of polynomials of the form $m'p$ where m' is a monomial, $p \in P$ and the degree of $m'p$ is at most d. From a practical point of view, the concept of mutants can be applied to the linear algebra step in the matrix-based algorithms for solving systems of multivariate polynomial equations, for example F4 and XL. In such algorithms, during the linear algebra step, the new polynomials that appear having a lower degree are mutants.

Applications of Mutants

In this section, we discuss how mutants have been and can be applied to different strategies and algorithms. After the description of the concept of mutant by Ding, some application algorithms have appeared. In SCC 2008, the MutantXL [55] algorithm and MutantF4 [56] algorithm were introduced in two papers. The application of mutants and its impact on the XL algorithm were the idea of the MutantXL algorithm while the application to the F4 algorithm was the case of MutantF4.

An improvement of MutantXL algorithm called MXL2 [126] was presented in PQCrypto 2008. In this thesis, we consider only the MutantXL and MXL2 which are our main contributions in this chapter. The MXL3 [123] algorithm is an improvement of MXL2 that computes a Gröbner basis instead of computing a unique solution as in MXL2. An intermediate version between MXL2 and MXL3, called MutantXL, was presented in [24]. In this thesis, we always mean by MutantXL, the original version which was presented at SCC 2008. The MGB [23] algorithm is an improvement of MXL3 that improves the selection strategy of MXL3 and MXL2 as well. The combination of

the concept of mutant and the sparse linear algebra solver based on the Wiedemann algorithm is introduced in the WMXL algorithm that is presented in this thesis in Chapter 6.

Border basis was introduced in [104] as a generalization of Gröbner basis. The main step in the computation of Border basis is the computation of a vector space basis. The concept of mutants can be used as an efficient tool to make the computation of vector space basis faster. Actually, the impact of mutants is to generate more polynomials at a lower degree which enable to compute vector space basis at a lower degree. This work is still in progress by Ehsan Ullah as a part of his PhD thesis under the supervision of Martin Kreutzer. In a private communication with Ehsan Ullah, we came to know that this work got some progress very recently. Ehsan concluded that, in case of $\mathbb{F}_q$ mutant strategy has almost the same effect for Border basis as it has for the XL algorithm and Gröbner basis. This means that further investigations in this area could lead towards some more improvements in computing Border basis.

Inspired by the idea of mutants, Ding and Schmidt [62], presented the Mutant Zhuang-Zi algorithm as an improvement to the original Zhuang-Zi (ZZ) algorithm. A survey about the problem of solving systems of multivariate polynomial equations with the focus to the MXL2 algorithm was presented in [146]. Another survey of different techniques in algebraic cryptanalysis with a discussion of the MutantXL attack was presented in [111]. An attempt to understand XL and MutantXL from a theoretical point of view was recently introduced in [164]. The complexity analysis of the Mutant-based algorithms was also discussed in [124].

4.3 The MutantXL Algorithm

MutantXL is an algorithm for solving systems of Boolean multivariate polynomial equations that was introduced at SCC 2008. MutantXL is an algorithm which is followed by a series of algorithms that rely primarily on the improvement of the XL algorithm.

In this section, we explain the MutantXL algorithm and how this algorithm is different from XL. We use the notations of Section 4.2. So P is a finite set of polynomials in R and we consider the system $P(x) = 0$ of multivariate polynomial equations.

Given a current degree D, as in XL the MutantXL algorithm extends the system of polynomial equations $P(x) = 0$ by multiplying the polynomials by all monomials in R up to degree $D - \max\{\deg(p) : p \in P\}$. Then, the system is linearized by considering the monomials as new variables and applying Gaussian elimination on the resulting linearized system. Afterwards, MutantXL searches for univariate equations. If no such equations exist, unlike XL, MutantXL searches for mutants that are new polynomials of degree less than current degree. If mutants are found, they are multiplied by all monomials such that the produced polynomials have degree $\leq D$. Using this strategy, MutantXL achieves to enlarge the system without incrementing D. If there are no mutants, like XL, MutantXL extends the initial polynomials to $D + 1$.

To simplify our explanation, we assume that the system $P(x) = 0$ is quadratic and has a unique solution. The main operations in the MutantXL algorithm are **Initialization**, **Elimination**, **Solution**, **Extraction** and **Extension**. The following is a description for each operation.

- **Initialization**:
  The set P is a container that includes all the generated polynomials during the algorithm. This set is initialized to the initial quadratic polynomial equations. The current degree,

D, is initialized to two. The degree at which we start the elimination step is called the elimination degree, eD. This degree is also initialized to two, since we assume that all the initial polynomials have the same degree. The set of mutants, M is initialize to the empty set.

- **Elimination**:
  The polynomials in P are linearized by replacing each monomial by a new variable. A coefficient matrix for the linearized system is constructed. The rows of this matrix represents the polynomials. The columns represents the monomials which are ordered with a graded lexicographic ordering. The Row Echelon Form (REF) of the coefficient matrix is computed.

- **Solution**:
  According to the used monomial ordering, all the initial variables followed by the constant are listed in the last columns of the coefficient matrix. After computing the REF, we search for univariate polynomials, $x_i + b_i$; where x_i is a variable and b_i is the corresponding value, 1 or 0, of that variable. In the case that all the corresponding values for all variables are found then MutantXL is terminated and returns the solution. In the case that only some variables have their corresponding values, we substitute these values in the set P. Afterwards, the value of D is modified to the maximum degree in P. The eD is assigned to the minimum degree in P.

- **Extraction**:
  If there exist some new polynomials of degree less than the current degree D, then these polynomials are mutants. These mutants are moved from P to M. All the lower degree mutants in M are multiplied by all monomials of degree $D - d$ where d is the degree of lower degree mutants. The new produced polynomials are added to P. The degree d mutants are removed from M. The elimination degree eD is assigned to $d + 1$. All the multiplications are reduced modulo the field equations.

- **Extension**:
  The current degree D is incremented by one. The polynomials in P are multiplied by all the monomials of degree $D - \deg(p), p \in P$. The new produced polynomials are added to the whole system P. All the multiplications are reduced modulo the field equations.

The MutantXL algorithm was introduced in [55] as Algorithm 4.1. The XL algorithm can be obtained from MutantXL if the **Extraction** step, the part in which mutants are found, multiplied, and added to the set of all polynomials, is skipped. See Figure 4.2 for a comparison with Figure 4.1.

In the MutantXL, the multiplication of the extracted mutants by all monomials at some degree D leads to new equations without generating new monomials. In most instances of the MQ-problem, these new equations help to solve these instances at degree D rather than increasing that degree as in XL. Indeed, solving systems of polynomial equations at a lower degree than XL is the main advantage of the MutantXL.

The main disadvantage of MutantXL as well as XL is the existence of trivial redundant polynomials which are reduced to zero in Gaussian elimination. These redundant polynomials are

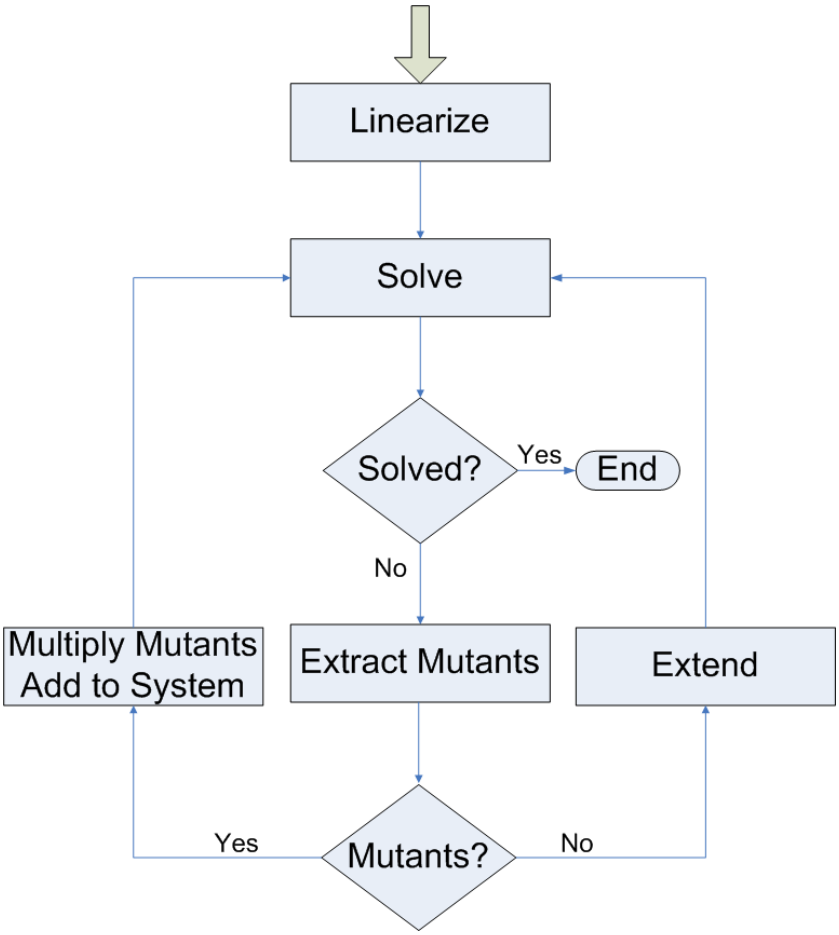

Figure 4.2: MutantXL

Algorithm 4.1 MutantXL

1: **Initialization**: Set the initial polynomials to *P*. Set the current degree $D = \max\{\deg(p) : p \in P\}$. Set the elimination degree $eD = \min\{\deg(p) : p \in P\}$. The set of mutants *M* is initialized to the empty set.
2: **Elimination**: Linearize *P*. Compute the row echelon form, $\tilde{P}$, of *P* up to degree $\leq eD$. Set $P = \tilde{P}$.
3: **Solution**: If there are univariate polynomials in *P*, then determine the values of the corresponding variables. If this solves the system return the solution and terminate. Otherwise, substitute the values for the variables in *P*. Set $D = \max\{\deg(p) : p \in P\}$ and set $eD = \min\{\deg(p) : p \in P\}$. Go back to **Elimination**.
4: **Extraction**: No univariate polynomials have been found in the previous step. Add all new polynomials of degree less than D in *P* to *M*. If $M \neq \emptyset$, then multiply all $m \in M$ of degree d where $d = \min\{\deg(m) : m \in M\}$ by all monomials such that the produced polynomials have a degree $\leq D$. Add the new polynomials obtained from the multiplication to *P*. Remove the multiplied polynomials from *M*. Set $eD = d + 1$. Go back to **Elimination**.
5: **Extension**: No mutants have been found in the previous step. Increment D by one. Multiply all polynomials in P by all monomials such that the produced polynomials have a degree $\leq D$. Set $eD = D$. Go back to **Elimination**.

generated during the process of multiplying polynomials by monomials. This multiplication is achieved either in the **Extraction** or the **Extension** steps. For example, let $p \in P$ be an initial quadratic polynomial. Let $x_i p$ be a polynomial in the extended system at degree three. In order to extend the whole system up to degree four, we multiply p by all quadratic monomials and multiply $x_i p$ by all variables. Therefore, multiplying p by $x_i x_j$ is equivalent to the multiplication of $x_i p$ by x_j.

4.4 MutantXL Toy Example

Consider the problem of solving the HFE system of 5 multivariate quadratic polynomial equations in 5 variables over $\mathbb{F}_2[x_1, x_2, x_3, x_4, x_5]$:

$$\begin{cases} p_1 : x_1x_3 + x_1x_4 + x_1x_5 + x_2x_3 + x_2x_4 + x_2x_5 + x_3x_4 + x_3x_5 + x_1 + x_2 = 0 \\ p_2 : x_1x_3 + x_1x_4 + x_1x_5 + x_2x_3 + x_2x_4 + x_2x_5 + x_3x_4 + x_3x_5 + x_3 + 1 = 0 \\ p_3 : x_1x_4 + x_1x_5 + x_2x_4 + x_2x_5 + x_3x_4 + x_3x_5 + x_1 + x_2 + x_3 = 0 \\ p_4 : x_1x_2 + x_1x_5 + x_2x_3 + x_2x_5 + x_3x_5 + x_2 + 1 = 0 \\ p_5 : x_1x_2 + x_2x_3 + x_2 = 0 \end{cases}$$

We start with degree D = 2, reduce the equations using Gaussian elimination. The reduced equations are

$$\begin{cases} p_4 : x_1x_2 + x_1x_5 + x_2x_3 + x_2x_5 + x_3x_5 + x_2 + 1 = 0 \\ p_6 : x_1x_3 + x_1x_4 + x_1x_5 + x_2x_3 + x_2x_4 + x_2x_5 + x_3x_4 + x_3x_5 + x_3 + 1 = 0 \\ p_7 : x_1x_4 + x_1x_5 + x_2x_4 + x_2x_5 + x_3x_4 + x_3x_5 + x_1 + x_2 + x_3 = 0 \\ p_8 : x_1x_5 + x_2x_5 + x_3x_5 + 1 = 0 \\ p_9 : x_1 + x_2 + x_3 + 1 = 0 \end{cases}$$

Only p_9 has degree $1 < D$ and there is no univariate equation. So we consider p_9 as a mutant, multiply it by all variables x_1, x_2, x_3, x_4, x_5 to obtain 5 new quadratic equations

$$\begin{cases} p_{10} : x_1x_2 + x_1x_3 = 0 \\ p_{11} : x_1x_2 + x_2x_3 = 0 \\ p_{12} : x_1x_3 + x_2x_3 = 0 \\ p_{13} : x_1x_4 + x_2x_4 + x_3x_4 + x_4 = 0 \\ p_{14} : x_1x_5 + x_2x_5 + x_3x_5 + x_5 = 0 \end{cases}$$

We add these equations to the eliminated equations obtained from previous Gaussian elimination. After new round of Gaussian elimination, we obtain the following new equations

$$\begin{cases} p_{15} : x_1 + x_2 + x_3 + 1 = 0 \\ p_{16} : x_2 = 0 \\ p_{17} : x_3 = 0 \\ p_{18} : x_4 = 0 \\ p_{19} : x_5 + 1 = 0 \end{cases}$$

Here p_{16}, p_{17}, p_{18} and p_{19} are univariate equations. After substitution, we obtain the following new equation

$$\{ \ p_{20} : x_1 + 1 = 0$$

which is univariate and solves the system. So MutantXL solves this system at D = 2, whereas XL solves this system at D = 3, since XL updates D by 1 directly after first elimination.

4.5 MutantXL Experimental Results

In this section, we present experimental results and compare the performance of MutantXL with the performance of XL. Our implementation uses the M4RI package which is released on May 2009. The MutantXL algorithm has been implemented using C/C++ based on the implementation of the XL algorithm, see Section 3.6 for more details. All these experiments are done on a Sun X4440 server, with four "Quad-Core AMD Opteron™ Processor 8356" CPUs and 128GB of main memory. Each CPU is running at 2.3 GHz. In these experiments we use only one out of the 16 cores.

We use seven HFE examples from [147] and another HFE system (25 equations in 25 variables) from the Hotaru distribution [154] denoted by HFEseg(d, m, n) and HFEhot(d, m, n), respectively,

where d is the degree of univariate polynomials in the HFE cryptosystem, m is the number of polynomials, and n is the number of variables.

The results can be found in Table 4.1. For each example we denote the system type, either HFEseg(d, m, n) or HFEhot(d, m, n) by "Sys" in the first column. The maximum degree that is reached in the algorithm is denoted by "D". The number of rows and columns of the largest matrix are represented by "nrows" and "ncols", respectively. The rank of this matrix can be found in the "Rank" column. The total number of mutants that is extracted and used by MutantXL is represented by "#Mut" column. Table 4.1 clearly shows that in six cases MutantXL outperforms XL and in the other two cases the two algorithms do the same. These two cases are the instances HFEseg(96, 11, 11) and HFEseg(96, 13, 13). After computing the row echelon form for the linearized system at degree four, a number of mutants of different degrees are appeared. But, the number of linear mutants is equal to the number of variables. Therefore, the two systems are solved without the need for any mutant multiplications. In the case that there are no mutants are used for the solving process, we say that the "#Mut" is equal to 0. The instance HFEhot(96, 25, 25) shows the effectiveness of the use of mutants. By using 1919 mutants, MutantXL algorithm is able to solve this instance at degree four while XL do this at degree six.

In Table 4.2 we also present detailed results for the HFEhot(96, 25, 25) instance. In this instance we initially have $D = 2$. Whenever the system is extended using the $\mathbf{Extension}$ process in Algorithm 4.1 or $\mathbf{Multiply}$ in Algorithm 3.2, we say that MutantXL and XL enter a new step. The step number is represented in the first column. For each step, we present the elimination degree in "eD" column. The matrix dimensions and the rank of the linearized system is represented in "nrows", "ncols", and "Rank" columns, respectively. The total number of mutants produced by MutantXL for each degree can be found in the last column, "#Mut".

In Table 4.3 we also present details for all the polynomials found in the new system after Gaussian elimination. The total number of polynomials produced by MutantXL for each degree can be found in the last row.

In order to measure the efficiency of MutantXL, we use instances of quadratic dense random systems which are the hardest instances of the MQ-problem. The instances from number of variables equal to 10 to number of variables equal to 24 were generated by Courtois [43] while the instances from 25 variables to 30 variables were generated by ourselves. We use an improved implementation for MutantXL. This improvement is based on extending the system up to some degree D by multiplying polynomials of degree $D - 1$ by variables instead of multiplying the original polynomials by monomials of degree $D - 2$ such that each initial polynomial is multiplied by each variable only once. The improved implementation for MutantXL is a feedback implementation from the MXL2 algorithm, see next section.

In Table 4.4, the first column "Sys" denotes the system type, MQR as abbreviation for Multivariate Quadratic Random, followed by the number of variables, which is the same as the number of equations, of the initial system. The maximum degree that the system can reach is denoted by "D". The number of rows and columns of the largest linear system to which Gaussian elimination is applied are represented by "nrows" and "ncols", respectively. The rank of this matrix can be found in the "Rank" column. The number of mutants are represented in "# Mutants" columns. The used memory in Megabytes (MiB), Gigabytes (GB) and the execution time in Seconds (S), Minuets (M) and Hours (H) are represented by "Mem" and "Time", respectively.

Table 4.4 shows that MutantXL can solve up to a dense random system of 29 multivariate quadratic polynomial equations in 29 variables using 128GB RAM. The memory and time perfor-

Table 4.1: Performance of MutantXL versus XL

Sys	XL				MutantXL				
	D	nrows	ncols	Rank	D	nrows	ncols	Rnk	#Mut
HFEseg(96, 7, 7)	4	203	99	98	3	63	64	62	14
HFEseg(96, 10, 10)	4	560	386	385	3	210	176	175	10
HFEseg(96, 11, 11)	4	737	562	561	4	737	562	561	0
HFEseg(96, 13, 13)	4	1196	1093	1092	4	1196	1093	1092	0
HFEseg(96, 15, 15)	5	8520	4944	4943	4	2340	1941	1760	270
HFEseg(96, 17, 17)	5	14025	9402	9401	4	3349	3214	2737	255
HFEseg(96, 19, 19)	4	3628	5036	3306	3	2565	1160	1159	117
HFEhot(96, 25, 25)	6	373450	245506	245505	4	14219	15276	13750	1919

Table 4.2: MutantXL: Detailed steps for HFEhot(96, 25, 25)

Step	eD	nrows	ncols	Rank	#Mut
1	2	25	326	25	0
2	3	650	2626	650	0
3	4	8150	15276	7825	50
4	4	9075	15276	9075	200
5	4	14075	15276	13719	1669
6	2	14219	15276	13750	–

Table 4.3: MutantXL: Detailed polynomials for HFEhot(96, 25, 25)

Step	#Degree1	#Degree2	#Degree3	#Degree4
1	0	25	0	0
2	0	0	625	0
3	0	0	50	7125
4	0	0	200	1050
5	20	249	1400	2975
6	5	26	0	0
Total	25	300	2275	11150

mance for the improved MutantXL algorithm is better than in the case of the XL algorithm, see Section 3.7, Table 3.1. Most of the dense random systems are solved at the same degree for both MutantXL and XL. We say that the "#Mut" is equal to 0 whenever the number of linear mutants is equal to the number of variables. Therefore, a system is solved without the help of mutants multiplication. The instances MQR14 and MQR15 are solved by the help of mutants at degree four using MutantXL while XL solves them at degree five. The instances MQR21, MQR22 and MQR23 are solved at degree five using MutantXL while XL solves them at degree six.

Even if there is no mutants, the performance of the improved MutantXL is better than XL. The reason for that performance is due to the way that is used to extend the systems up to some degree. The improved MutantXL avoids some redundant polynomials. As an outcome, the number of polynomials generated by XL is greater than in the case of MutantXL.

4.6 The MXL2 Algorithm

MXL2 is an algorithm for solving systems of Boolean multivariate polynomial equations that was proposed at PQCrypto 2008. It is an improvement of MutantXL which in turn improves the XL algorithm. The MXL2 and MutantXL algorithms are similar in using the concept of mutants that is introduced in Section 4.2, while MXL2 uses two substantial improvements over $\mathbb{F}_2$ that oftentimes allows to solve systems with significantly smaller matrix sizes than XL and MutantXL. Moreover, MXL2 multiplies the lower degree mutants by monomials of degree one instead of multiplying by monomials such that the produced polynomials have a degree less than or equal to some bounded degree. In this section we explain the MXL2 improvements, then introduce the algorithm.

Table 4.4: Performance of MutantXL for Determined Dense Random Systems

Sys	D	nrows	ncols	Rank	#Mutants	Mem	Time
MQR10	4	560	386	385	0	3MiB	1S
MQR11	4	737	562	561	0	3MiB	1S
MQR12	4	948	794	793	0	5MiB	1S
MQR13	4	1196	1093	1092	0	4MiB	1S
MQR14	4	1771	1471	1470	196	4MiB	1S
MQR15	4	3045	1941	1940	90	4MiB	1S
MQR16	5	9447	6885	6884	0	17MiB	1S
MQR17	5	12118	9402	9401	0	28MiB	2S
MQR18	5	15340	12616	12615	0	46MiB	4S
MQR19	5	19193	16664	16663	0	77MiB	7S
MQR20	5	23708	21700	21699	0	124MiB	12S
MQR21	5	28455	27896	27895	2730	196MiB	21S
MQR22	5	77209	35443	35442	1980	490MiB	4M
MQR23	5	57408	44552	44551	736	557MiB	5M
MQR24	6	231581	190051	190050	0	9.3GB	1.2H
MQR25	6	287992	245506	245504	0	15.3GB	2.2H
MQR26	6	355063	313912	313911	0	24.5GB	3.6H
MQR27	6	436578	397594	397593	0	38.6GB	6.6H
MQR28	6	528911	499178	499177	0	59.7GB	10.7H
MQR29	6	637173	621616	621615	0	91.1GB	16.2H

In many experiments with MutantXL on some HFE systems and some randomly generated multivariate quadratic systems, we noticed that there are two problems. The first occurs when the number of lower degree mutants is very large. We observed that the multiplication of this number of mutants produces many reductions to zero. A second problem occurs when an iteration does not produce mutants at all or produces only an insufficient number of mutants to solve the system at lower degree D. In this case MutantXL behaves like XL.

Our proposed improvement handles these two problems, while using the same linearization strategy as the original MutantXL. Moreover, the solution is achieved with a lower number of polynomials. In particular, MXL2 has two important advantages over MutantXL. The first is the use of the necessary number of mutants and the second is extending the system only partially to higher degrees.

The main idea for the first improvement is to multiply only a necessary number of mutants to the given system. This number is numerically computed. In order to explain this improvement in detail, we need the following notations as well as the notations of Section 4.2.

Let $S_k = \{x_1^{k_1} \cdot x_2^{k_2} \dots x_n^{k_n} : k_i \in \{0,1\}, k_1 + \dots + k_n \leq k\}$ be the set of all monomials that have degree less than or equal to k. Combinatorially, the number of elements of this set can be computed as

$$|S_k| = \sum_{\ell=1}^{k} \binom{n}{\ell}, 1 \leq k \leq n \tag{4.1}$$

where n is the number of variables.

Let k be the degree of the lowest-degree mutant occurring and the number of the linearly independent elements of degree $\leq k+1$ in P be $Q(k+1)$. Then the smallest number of mutants that are needed to generate $|S_{k+1}|$ linearly independent equations of degree $\leq k+1$ is

$$\lceil (|S_{k+1}| - Q(k+1))/n \rceil, \tag{4.2}$$

where S_{k+1} is as in (4.1) and n is the number of variables.

The necessary number of mutants could be illustrated by Figure 4.3. At some degree $k+1$ after Gaussian elimination, we could find degree $k+1$ polynomials, some old degree k polynomials and some new degree k polynomials which are mutants. In order to obtain a full rank matrix at degree $k+1$, the number of monomials which is $|S_{k+1}|$ should be equal to the number of linearly independent polynomials. If we have $Q(k+1)$ linearly independent elements of degree $\leq k+1$, then the difference between $|S_{k+1}|$ and $Q(k+1)$ is the missing number of polynomials. Every mutant is multiplied by all n variables, then we should divide the number of missing polynomials, $|S_{k+1}| - Q(k+1)$, by n to obtain the necessary number of mutants to be multiplied.

By using not all the emerged mutants, the efficiency is increased. For space efficiency only a few mutants are used and for the time efficiency the multiplications to generate higher degree polynomials do not have to be performed to all mutants. Therefore by multiplying only the necessary number of mutants, the system can potentially be solved by a smaller number of polynomials and a minimum number of multiplications. This handles the first problem.

In the following we explain how MXL2 solves the second problem. The solution of this problem which is based on the so-called partial enlargement strategy is due to Mohamed Saied Emam Mohamed. Suppose we have a system without enough mutants. In this case, we noticed that in the process of space enlargement, MutantXL multiplies all original polynomials by all monomials

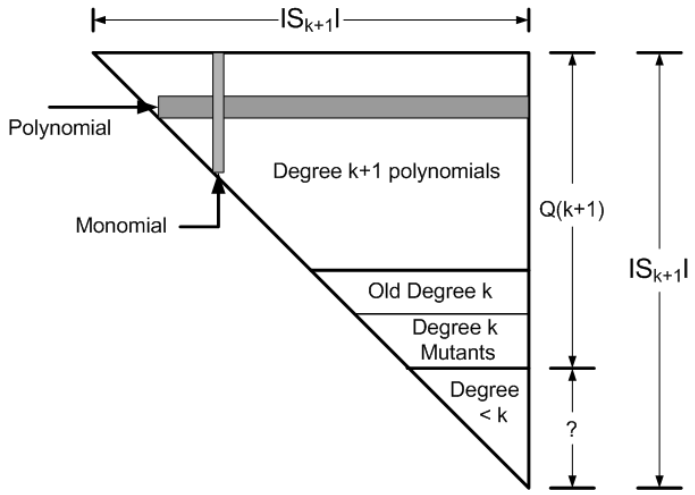

Figure 4.3: Necessary Number of Mutants

of degree $D - 2$. In most cases, only a small number of extended polynomials that are produced are needed to solve the system. Moreover, the system will be solved only when some of these elements are reduced to lower degree elements. To be more precise, the degree of the extended polynomials is decreased only if the higher degree terms are eliminated. We have found that by using a partial enlargement strategy and a successive multiplication of polynomials with variables, while excluding redundant products, we can solve the system with a smaller number of equations. To discuss this idea in details we first need to define the following:

Definition 4.2. *The* leading variable *of a polynomial p in R is x, if x is the smallest variable, according to the order defined on the variables, in the leading term of p. It can be written as*

$$\mathrm{LV}(p) = x \tag{4.3}$$

Definition 4.3. *Let $P_k = \{p \in P\text{: } deg(p) = k\}$ and $x \in X$. We define P_k^x as follows*

$$P_k^x = \{p \in P_k : LV(p) = x\} \tag{4.4}$$

In the process of space enlargement, MXL2 deals with the polynomials of P_D differently. Let P_D be divided into a set of subsets depending on the leading variable of each polynomial in it. In other words, $P_D = \bigcup_{x \in X} P_D^x$, where X is the set of variables as defined previously and P_D^x as in (4.4). MXL2 enlarges P by incrementing D and multiplying the elements of P_D as follows: Let x be the largest variable, according to the order defined on the variables, that has $P_D^x \neq \emptyset$. MXL2 successively multiplies each polynomial of P_D^x by variables such that each variable is multiplied only once. This process is repeated for the next smaller variable x with $P_D^x \neq \emptyset$ until the solution is obtained, otherwise the system is enlarged to the next D. Therefore MXL2 may solve the system by enlarging only subsets of P_D, while MutantXL solves the system by enlarging all the elements of P_D. MXL2 handles the second problem by using this partial enlargement strategy.

In other words, the second improvement is the usage of the partial enlargement strategy in which the polynomials at degree D are divided according to their leading variable. Instead of enlarging all the partitions, only the non-empty partitions are multiplied by all the variables that are smaller than the previous multiplied variable. This is accomplished partition by partition.

This partial enlargement strategy gives also an improvement in time and space since the system can be solved using a lower number of polynomials. Moreover, in some systems mutants may appear in the last step together with the solution of the system. These mutants are not fully utilized. Using partial enlargement strategy enforces these mutants to appear before the system is solved.

As a result of using the two MXL2 improvements, MXL2 generates the same solution as if all mutants would have been used and all partitions would have been multiplied. MXL2 solves multivariate polynomial equations using a smaller number of polynomials than MutantXL. We show that the system is partially enlarged, so MXL2 leads to the original MutantXL if the system is solved with enlarging the last partition. As an outcome, MXL2 outperforms the original MutantXL if it solves the system by earlier partition enlarged. This will be clarified experimentally in Section 4.7.

At this point, we can define the MXL2 algorithm. We use the notations of Section 4.2. For simplicity as we did for MutantXL, we assume that the system of multivariate polynomial equations is quadratic and has a unique solution. Moreover, the set of solutions of the system is defined as $\{x = b : x \text{ is variable and } b \in \{0, 1\}\}$. First of all, we give a brief description of the variables that are used during the algorithm. After that, we present the MXL2 algorithm. This is followed by a formal description to MXL2. Throughout the whole algorithm, we use the graded lexicographical ordering on the variables.

In order to multiply a polynomial by variables instead of monomials such that each variable is multiplied only once, we use a one dimension array, called *History*, in which the variable multiplier for each polynomial is stored. For example, if we multiply p_1 by x_2 then *History*[1] = 2. For the original polynomials the variable multiplier is 0. In order to initialize all the elements of *History* with zeros, we denote this by SetArray(*History*, | *N* |, 0), where |N| is the number of elements in *History*. The set *Roots* denotes all the polynomials of degree less than or equal to 2. The set *M* denotes all the extracted mutants. The set *P* denotes a container to all the polynomials that appeared during the algorithm. The Boolean variable *eXtended* denotes a flag to determine whether a system is extended to the next degree or not. The degree bound of all elements in *P* is denoted by D. eD denotes the elimination degree that is used to eliminate a subset of *P*. The MXL2 algorithm was presented in [126] as Algorithm 4.2.

To give a more formal description of the MXL2 algorithm and its sub-algorithms, firstly we need to define the following subroutines:

- **Solve**($Roots, X$): if there are univariate equations in the $Roots$, then solve them and return the solutions.
- **Substitute**($Solution, Roots$): use all the solutions found to simplify the $Roots$.
- **SetArray**($History, n, 0$): set a one dimensional array, $History$, to an array with number of elements equal to n and initialize these elements by zeros.
- **AppendArray**($History, n, 0$): append to $History$ an array with number of elements equal to n and initialize these elements by zeros.

Algorithm 4.2 MXL2

1: **Initialization**: Set the initial polynomials to *P*. Set the degree bound $D = \max\{\deg(p) : p \in P\}$. Set the elimination degree $eD = \min\{\deg(p) : p \in P\}$. The set of mutants *M* and the set of initial polynomials, *Roots*, are initialized to the empty set. Set the elements of *History*, an array of number of elements as *P*, to zeros, i.e. SetArray(*History*, $\mid P \mid$, 0). Set the boolean flag *eXtended* to false.

2: **Elimination**: Consider each monomial in *P* as a new variable i.e. Linearize *P*. Compute the row echelon form $\tilde{P}$ of *P* up to degree $\leq eD$. Set *P* = $\tilde{P}$.

3: **RootsExtraction**: Copy all new polynomials of degree ≤ 2 to the *Roots*.

4: **Solution**: If there are univariate polynomials in the *Roots*, then determine the values of the corresponding variables, and remove the solved variables from the variable set. If this solves the system return the solution and terminate. Otherwise, substitute the values for the variables in the *Roots*, set *P* to the *Roots*, set eD to the maximum degree of the *Roots*, SetArray(*History*, $\mid P \mid$, 0), and go back to **Elimination**.

5: **MutantsExtraction**: No univariate polynomials have been found in the previous step. Copy all new polynomials of degree $< D$ from *P* to *M*.

6: **Extension**: If $M \neq \emptyset$, then extend the *History* by an array of the number of elements of the same length as the new polynomials and initialize these new elements by zeros. Multiply the necessary number of mutants $m \in M$ with degree d where $d = \min\{\deg(m) : m \in M\}$ by all variables. Set the *History* for each new polynomial by its variable multiplier. Include the resulting polynomials in *P*. Set the eD to $d + 1$. Remove all multiplied mutants from *M*.
Otherwise, No mutants have been found. if *eXtended* is false; then increment D by 1, set x to the largest leading variable under the variable order satisfies that $P^x_{D-1} \neq \emptyset$, set *eXtended* flag to true. Multiply each polynomial p in P^x_{D-1} by all unsolved variables $<$ the variable stored in the *History* of p. Include the resulting polynomials in *P*. Set x to the next smaller leading variable satisfies that $P^x_{D-1} \neq \emptyset$, if there is no such variable, then set *eXtended* to false. eD to D. Go back to **Elimination**.

- **SelectNecessary**(M, D, k, n): compute the necessary number of mutants with degree k as in equation (4.2), let the mutants be ordered depending on their leading terms, then return the necessary mutants by ascending order.
- **Xpartition**(P, x): return $\{p \in P : LV(p) = x\}$.
- **LargestLeading**(P): return $\max\{y : y = LV(p), p \in P, y \in X\}$.
- **NextSmallerLeading**(P, x): return $\max\{y : y = LV(p), p \in P, y \in X \text{ and } y < x\}$.

Algorithm 4.3 represents the formal description of MXL2. This is followed by Algorithm 4.4 which is responsible to enlarge the subset of the equations to a higher degree. Then, Algorithm 4.5 is introduced which is responsible for extracting mutants.

4.7 MXL2 Experimental Results

In this section, we present the experimental results for our implementation of the MXL2 algorithm. We compare MXL2 with the original MutantXL, Magma's implementation of F4, and the XL algorithm for some random systems (5-24 equations in 5-24 variables). The results can be found in Table 4.5. Moreover, we have another comparison for MXL2, original MutantXL, and Magma's F4 for some HFE systems (25-55 equations in 25-55 variables) in order to clarify that the mutant strategy has the ability to be helpful with different types of systems. See the results in Table 4.6.

Random systems are taken from [43]. These systems are denoted by MQR, followed by the number of variables and equations. The HFE systems (30-55 equations in 30-55 variables) are generated with code contained in [147]. We denote these systems by HFEseg(d, m, n), where d is the degree of univariate polynomial in the HFE cryptosystem, m is the number of equations which is equal to the number of variables n. The HFE system (25 equations in 25 variables) was taken from the Hotaru distribution [154], denoted by HFEhot(d, m, n).

The results for F4 are obtained using Magma version 2.13-10 released on January 2007 ; the parameter `HFE:=true` was used to solve HFE systems. For each example, we present the system type either HFEseg(d, m, n) or HFEhot(d, m, n) by "Sys" column. The size of the largest linear system, to which Gaussian elimination is applied, is represented under each method. The '*' in the first column for random systems means that we use mutants to solve such systems.

In all experiments, the highest degree of the polynomials generated by MutantXL and MXL2 is equal to the highest degree of the S-polynomial in F4. In the MXL2 implementation, we use only one matrix from the start to the end of the process by enlarging and extending the initial matrix, the largest matrix is the accumulative of all polynomials that are held in the memory.

In Table 4.5, we see that in practice MXL2 is an improvement for memory efficiency over the original MutantXL. Systems for which mutants are produced during the computation, MutantXL is better than XL. If no mutants occur, MutantXL behaves identically to XL. Comparing XL, MutantXL, and MXL2; MXL2 is the most efficient even if there are no mutants. In almost all cases MXL2 has the smallest number of columns as well as a smaller number of rows compared to the F4 implementation contained in Magma. We can see easily that 70% of the cases MXL2 is better, 5% is equal, and 25% is worse.

In Table 4.6, we also present a comparison based on HFE systems . In all these seven examples for all the three algorithms (Magma's F4, MutantXL, and MXL2), all the monomials up to degree

Algorithm 4.3 Formal MXL2

Inputs
  F: set of quadratic polynomials with a unique solution.
  X: set of variables.
Output
  $Solution$: The solution of $F = 0$.
Variables
  RP: set of all polynomials produced during the process.
  M: set of mutants.
  $Roots$: set of all polynomials of degree ≤ 2.
  x: a variable.
  D: current system degree starts by 2.
  eD: elimination degree.
  $history$: array of length $|RP|$ to store previous variable multiplier.
  $eXtended$: a flag to enlarge the system.
Begin
Initialization()
$\{RP \leftarrow F$; $M, Solution \leftarrow \varnothing$; $D \leftarrow 2$; $eD \leftarrow 2$; SetArray($history, |RP|, 1$); $eXtended \leftarrow$ FALSE $\}$
repeat
  Linearize RP using graded lex order.
  Eliminate(Extract($RP, eD, \leq$), $history$).
  $Roots \leftarrow Roots \cup$ Extract($RP, 2, \leq$)
  $Solution \leftarrow Solution \cup$ Solve($Roots, X$)
  if $|Solution| > 0$ **then**
    $Roots \leftarrow$ Substitute($Solution, Roots$)
    $RP \leftarrow Roots$
    $history \leftarrow$ SetArray($history, |Roots|, 1$)
    $M \leftarrow \emptyset$
    $eD \leftarrow D \leftarrow$ max$\{deg(p) : p \in Roots\}$
  else
    $M \leftarrow M \cup$ Extract($RP, D - 1, \leq$)
    $RP \leftarrow RP \cup$ Enlarge($RP, M, X, D, x, history, eXtended, eD$)
  end if
until $Roots = \varnothing$
End

Algorithm 4.4 Enlarge($RP, M, X, D, x, history, eXtended, eD$)

$history, eXtended, eD$: may be changed during the process.
Variables
  NP: set of new polynomials.
  NM: set of necessary mutants.
  Q: set of degree $D-1$ polynomials which have the leading variable x.
  k: minimum degree of the mutants.
Begin
$NP \leftarrow \emptyset$
if $M \neq \emptyset$ **then**
  $k \leftarrow \min\{deg(p) : p \in M\}$
  $NM \leftarrow$ SelectNecessary($M, D, k, |X|$)
  AppendArray($history, |X| \cdot |NM|, 1$)
  for all $p \in NM$ **do**
    for all y in X **do**
      $NP \leftarrow NP \cup \{y \cdot p\}$
      $history[y \cdot p] = y$
    end for
  end for
  $M \leftarrow M \setminus NM$
  $eD \leftarrow k + 1$
else
  if not $eXtended$ **then**
    $D \leftarrow D + 1$
    $x \leftarrow$ LargestLeading(Extract($RP, D-1, =$))
    $eXtended \leftarrow$ TRUE
  end if
  $Q \leftarrow$ XPartition(Extract($RP, D-1, =$), x)
  Extend($history, |X| \cdot |Q|$)
  for all $p \in Q$ **do**
    for all $y \in X$: $y < history[p]$ **do**
      $NP \leftarrow NP \cup \{y \cdot p\}$
      $history[y \cdot p] \leftarrow y$
    end for
  end for
  $x \leftarrow$ NextSmallerLeading(Extract($RP, D-1, =$), x)
  if x = smallest variable **then**
    $eXtended \leftarrow$ FALSE
  end if
  $eD \leftarrow D$
end if
Return NP
End

Algorithm 4.5 Extract($P, degree, operation$)

P: set of polynomials.
SP: set of selected polynomials.
$operation$: conditional operation that belongs to $\{<, \leq, >, \geq, =\}$.
Begin
$SP \leftarrow \varnothing$
for all $p \in P$ **do**
  if $\deg(p)$ $operation$ $degree$ **then**
    $SP \leftarrow SP \cup \{p\}$
  end if
end for
Return SP
End

Table 4.5: Matrix Dimensions for Dense Random Systems Comparison

Sys	XL	MutantXL	$F4_{v2.13}$	MXL2
MQR5	30×26	30×26	30×26	**20×25**
MQR6*	42×42	47×42	46×40	**33×38**
MQR7*	203×99	154×64	154×64	**63×64**
MQR8*	296×163	136×93	131×88	**96×93**
MQR9	414×256	414×256	480×226	**151×149**
MQR10	560×386	560×386	624×3396	**228×281**
MQR11	737×562	737×562	804×503	**408×423**
MQR12	948×794	948×794	1005×704	**519×610**
MQR13	1196×1093	1196×1093	1251×980	**1096×927**
MQR14*	6475×3473	1771×1471	1538×1336	**1191×1185**
MQR15*	8520×4944	3045×2941	2639×1535	**1946×1758**
MQR16	11016×6885	9447×6885	9993×4034	**2840×2861**
MQR17	14025×9402	12118×9402	12382×5784	**3740×4184**
MQR18	17613×12616	15340×12616	15187×8120	**6508×7043**
MQR19	21850×16664	19193×16664	18441×11041	**9185×11212**
MQR20	26810×21700	23708×21700	22441×14979	**14302×12384**
MQR21*	153405×82160	28970×27896	26860×19756	**14365×20945**
MQR22*	194579×110056	77209×35443	63621×21855	**35463×25342**
MQR23*	244145×145499	57408×44552	41866×29010	**39263×36343**
MQR24	303324×190051	231581×190051	207150×78637	**75825×69708**

Table 4.6: Matrix Dimensions for HFE Systems Comparison

Sys	MutantXL	$F4_{v2.13}$	MXL2
HFEhot(96, 25, 25)	14218×15276	12495×15276	**11926×15276**
HFEseg(64, 30, 30)	26922×31931	23832×31931	**19174×31931**
HFEseg(48, 35, 35)	31255×59536	27644×59536	**30030×59536**
HFEseg(33, 40, 40)	49620×102091	45210×102091	**46693×102091**
HFEseg(24, 45, 45)	57734×164221	43575×164221	**45480×164221**
HFEseg(40, 50, 50)	85025×251176	75012×251176	**67826×251176**
HFEseg(48, 55, 55)	119515×368831	104068×368831	**60116×368831**

bound D appear in Magma, MutantXL, and MXL2. Therefore, the number of columns is the same in all the three algorithms. It is clear that MXL2 has a smaller number of rows in four cases of seven. In all cases MXL2 outperforms MutantXL.

A time comparison in seconds for dense random systems between MutantXL and MXL2 can be found in Figure 4.4. In this comparison, we use a Sun X2200 M2 server with 2 dual core Opteron 2218 CPU running at 2.6GHz and 8GB of RAM which was so-called Userver1 that belongs to the Cryptography and Computer Algebra group, Computer Science Department, TU-Darmstadt. From Figure 4.4, it is clear that the MXL2 has a good performance for speed compared to MutantXL.

In order to shed light on which strategy (necessary mutants or partial enlargement) worked more than the other in which case, we make another comparison for dense random systems. In this comparison, we have 4 methods that cover all possibilities to use the two strategies. Method1 is for multiplying all lower degree mutants that are extracted at certain level: none of the two strategies are used. Method2 is for multiplying only our claimed necessary number of mutants, necessary mutant strategy. We use Method3 for partial enlargement strategy, multiplications are for all lower degree mutants. For both strategies which is MXL2, we use Method4. See Table 4.7.

In Table 4.7, comparing Method1 and Method2, we see that practically the strategy of necessary number of mutants sometimes has an effect in the cases which have a large enough number of mutants (cases MQR7, MQR8, MQR14, MQR15, MQR22 and MQR23). In the case when there are less mutants (cases MQR6, MQR21 and MQR24) or no mutants at all (cases MQR5, MQR9, MQR10-MQR13, and MQR16-MQR20), the total number of rows is the same as in Method1. Furthermore, in case MQR22 because not all mutants were multiplied, the number of columns is decreased. By comparing Method1 and Method3, in most of the cases for the partial enlargement strategy, we have a smaller number of rows except for case MQR13 which is worse because Method3 extracts mutants earlier than Method1, so it multiplies all these mutants while MutantXL solves and ends before multiplying them. In the case that is solved with the last partition, the two methods are identical (case MQR7 and MQR8).

Indeed, using both strategies as in Method4 is the best choice. In all cases the number of rows in this method is less than or equal the minimum number of rows for both Method2 and Method3,

$$\#rows\ in\ Method4 \leq min(\#rows\ in\ Method2,\ \#rows\ in\ Method3)$$

In some cases (MQR13, MQR15 and MQR22) using both strategies leads to a smaller number of columns.

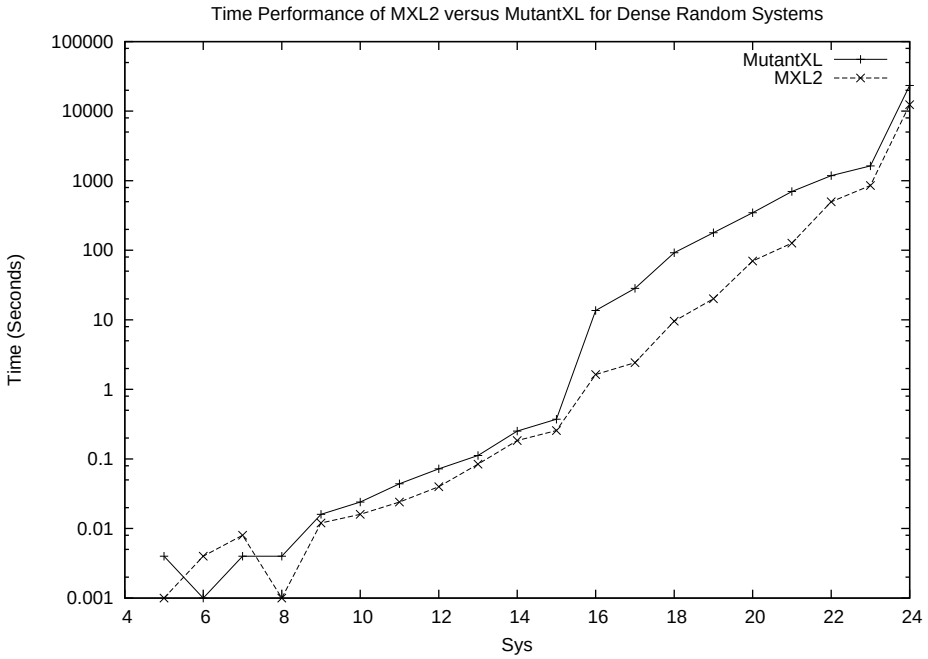

Figure 4.4: Time Performance of MXL2 versus MutantXL for Dense Random Systems

Table 4.7: Matrix Dimensions for Comparing Different improvement Strategies in MXL2

Sys	Method1	Method2	Method3	Method4
MQR5	30×26	30×26	25×25	**20×25**
MQR6	47×42	47×42	33×38	**33×38**
MQR7	154×64	63×64	154×64	**63×64**
MQR8	136×93	96×93	136×93	**96×93**
MQR9	414×239	414×239	232×149	**151×149**
MQR10	560×367	560×367	318×281	**228×281**
MQR11	737×541	737×541	408×423	**408×423**
MQR12	948×771	948×771	519×610	**519×610**
MQR13	1196×1068	1196×1068	1616×967	**1096×927**
MQR14	1771×1444	1484×1444	1485×1185	**1191×1185**
MQR15	2786×1921	1946×1921	2681×1807	**1946×1758**
MQR16	11016×5592	10681×5592	6552×2861	**2840×2861**
MQR17	14025×7919	13601×7919	4862×4184	**3740×4184**
MQR18	17613×10930	17086×10930	6508×7043	**6508×7043**
MQR19	21850×14762	21205×14762	9185×11212	**9185×11212**
MQR20	26810×19554	26031×19554	14302×12384	**14302×12384**
MQR21	31641×25447	31641×25447	14428×20945	**14365×20945**
MQR22	92831×34624	38116×32665	56385×28195	**35463×25342**
MQR23	76558×43650	45541×43650	39263×36343	**39263×36343**
MQR24	298477×190051	297810×190051	75825×69708	**75825×69708**

We present experimental results comparing MXL2 to the XL algorithm, the MutantXL algorithm and F4 algorithm, implemented in Magma. For this comparison, we have chosen randomly generated instances of the MQ-problem and quadratic systems derived from HFE cryptosystem instances. In both cases, the largest matrices produced by MXL2 are substantially smaller than the ones produced by MutantXL and XL. Moreover, for a significant number of cases we even see a reduction of the size of the largest matrix when we compare MXL2 against Magma's implementation of F4.

4.8 Practical Applications of Mutant-based Algorithms

In order to give a value to a tool for solving systems of multivariate polynomial equations, we need to provide a real world applications to that tool. In this section, the real world applications of the MutantXL and MXL2 algorithms are explored. Efficiently, MutantXL was able to break the MQQ cryptosystem while MXL2 was able to break both the Little Dragon Two and Poly-Dragon cryptosystems more efficiently than the F4 algorithm. The practical algebraic cryptanalysis for both Little Dragon Two and Poly-Dragon was published in [22].

4.8.1 Cryptanalysis of MQQ Cryptosystems

In [125], an efficient practical attack of the Multivariate Quadratic Quasigroups (MQQ) cryptosystem by solving systems of multivariate quadratic polynomial equations using a modified version of the MutantXL algorithm is presented. This work was done by three of the mutant team members namely, Mohamed Saied Emam, Jintai Ding and Johanes Buchmann. An explanation of the reason that systems arising in MQQ are so easy to solve in practice can be found in [76]. A more detailed information about the MQQ cryptosystems that is based on multivariate quadratic quasigroups and a special transform called Dobbertin transformation can be found in [87].

4.8.2 Cryptanalysis of Dragon-based Cryptosystems

We refer to the Little Dragon Two (LD2) and the Poly-Dragon Cryptosystems, see Chapter 2, as Dragon-based cryptosystems. In this section we present experimental results of the attack on the LD2 and Poly-Dragon by using MXL2 and compare the performance of MXL2 with two versions of Magma's implementation of F4 namely V2.13-10 released on January 2007 and V2.16 released on November 2009. The reason for using these two versions is that when we used Magma's version (V2.16), we found that this version solves the LD2 and Poly-Dragon systems at degree four while MXL2 as well as Magma V2.13-10 solves at degree three. In this context, it is not fair to use only this version (V2.16) in the comparison. For both Magma versions, we use the graded lexicographic ordering and the field equations are included.

The main task for a cryptanalyst is to find a solution of the systems of equations that represent the LD2 and Poly-Dragon schemes. These systems are essentially implemented as described in [157] and [142], respectively. Magma version (2.16) has been used for the implementation by Fabian Werner. Due to the high number of variables, this direct approach is not very efficient but it is sufficient for modeling purposes. For real-life applications, there are more elegant ways to create the public key using specialized software and techniques like polynomial interpolation (see [176] for example).

Table 4.8: MXL2: Results for LD2-229Var

Round	eD	Matrix	Rank	#Mutants	#LM	#Uni
1	2	229×26336	229	0	0	0
2	3	457×1975812	457	0	0	0
3	3	52670×2001690	52669	686	228	2
4	2	915×26336	913	226	226	108
5	2	913×26336	805	118	118	0
6	2	14847×26336	7140	1	1	1
7	2	7140×26336	7021	118	118	118

All the experiments are done on a Sun X4440 server, with four "Quad-Core AMD Opteron™ Processor 8356" CPUs and 128GB of main memory. Each CPU is running at 2.3 GHz. In these experiments we used only one out of the 16 cores.

We try to solve different systems with the same number of variables. As a result of our experiments, we notice that the complexity for different systems of LD2 and Poly-Dragon schemes with the same number of variable will be, essentially, the same. In this context, the results given in this section are for one particular instance for each system.

Table 4.8 presents the required steps of solving an LD2 instance of $n = 229$ using MXL2. In this table, for each step (Round) we present the elimination degree (eD), the matrix dimensions (Matrix), the rank of the matrix (Rank), the total number of mutants found (#Mutants), the number of linear mutants found (#LM) and the number of univariate polynomials found (#Uni).

Table 4.9 and Table 4.10 show results of the LD2 systems for the range 79-229 equations in 79-229 variables and results of the Poly-Dragon systems for the range 79-299 equations in 79-299 variables, respectively. The first column "Sys" denotes the number of variables and the number of equations for each system. The highest degree of the elements of the system that occurred during the computation is denoted by "D". The used memory in Megabytes and the execution time in seconds can be found in the columns represented by "Mem" and "Time", respectively, except for bigger systems it is in hours (H) or days (D). In both tables we can see that MXL2 always outperforms both versions of Magma's F4 in terms of memory and time.

Table 4.11 shows the required rounds of solving an Poly-Dragon instance of $n = 259$ using MXL2. The columns there are the same as in Table 4.8. From Table 4.11 we can see that MXL2 solves the Poly-Dragon instance with 259 variables in 7 rounds. In the first round of the algorithm, there was no dependency in the original 259 polynomials and no mutants were found. Therefore, MXL2 extended the system partially to generate new 258 cubic equations. In the second round, after applying the **Elimination** step to the extended system (517 equations), all the equations were independent and there were no mutants found. The MXL2 extended the system again by applying **Extension** step to generate 66823 new cubic equations. By echelonizing the resulting extended system (67340 equations), we obtain a system of rank 67,339, a number of 518 quadratic mutants and a number of 258 linear mutants in which two equations are univariate. After simplifying the extended system with the two univariate polynomials and modifying the elimination degree to two, we obtain a quadratic system of (1035 equations). The third round is finished. In the fourth round, echelonizing the system of 1035 equations at degree two, yield a system of rank 1033 and 256 linear mutants, 138 out of them are univariate. Substituting with the 138 univariate equations

Table 4.9: Performance of MXL2 versus F4 for Little-Dragon-Two

Sys	$F4_{v2.13-10}$			$F4_{v2.16}$			MXL2		
	D	Mem	$Time$	D	Mem	$Time$	D	Mem	$Time$
79	3	490	29	4	321	26	3	211	22
89	3	841	116	4	1600	203	3	346	40
99	3	1357	238	4	2769	411	3	545	73
109	3	2092	500	4	2046	331	3	844	122
119	3	3102	998	4	6842	1142	3	1251	217
129	3	4479	1827	4	11541	2529	3	2380	458
139	3	6280	3134	4	8750	1723	3	3387	742
149	3	8602	4586	4	23325	5795	3	3490	692
159	3	11547	7466	4	30178	7845	3	4545	1146
169	3	15191	11478	4	46381	18551	3	6315	1613
179	3	19738	17134	4	46060	17502	3	8298	2025
189	3	25234	28263	4	91793	54655	3	10697	2635
199	3	31848	11.24H	4	134159	1.47D	3	13772	1H
209	3	39800	16.36H	4	97834	13.19H	3	17431	1.32H
219	3	49,134	1.11D	4	184,516	2.92D	3	29,856	2.81H
229	3	60,261	1.56D	Ran out of memory			3	25,847	2.60H

and eliminating, we obtain a system of rank 895 and 118 linear mutants in round 5. The necessary number of mutants that are required at this round is 250. Therefore, all the 118 linear mutants are multiplied by variables using **Extension**, multiply mutants part. In round 6, we obtain one linear mutant which is also univariate polynomial from eliminating the extended system of total 14937 equations and rank of 7140. In round 7, after substituting with the univariate equation, we start with 7140 equations and the elimination degree is the same, two. We obtain a matrix of rank 7021 and the rest 118 univariate equations after applying the **Elimination** step.

Table 4.12 shows the required steps of solving the same Poly-Dragon instance as in Table 4.11 using F4 version (V2.13.10). In each step, we show the step degree (sD), the matrix dimensions (Matrix) and the number of pairs (#Pairs).

Experimentally, as far as we notice, the main new feature of Magma's F4 version (V2.16) is that at a certain degree, the program is interrupted after generating a number of linear polynomials and then a new phase is started with extended basis by adding the linear polynomials to the original ones then compute a Gröbner basis of the extended new system. Figure 4.5 shows a comparison between the number of linear mutants that are generated at degree three and the number of linear polynomials generated by F4 (V2.16) at degree three for each LD2 system. These mutants as well as linear polynomials generated by F4 show that there is a hidden algebraic structure in the LD2 scheme that distinguishes it from a random system.

The generated number of linear polynomials at which Magma's F4 version (2.16) interrupts the first phase of computation is not enough to finish computing a Gröbner basis at the same degree at which these linear polynomials appear. The usage of the necessary number of mutants for MXL2 could help new Magma's F4 to recover its defect.

Table 4.10: Performance of MXL2 versus F4 for Poly-Dragon

Sys	$F4_{v2.13-10}$			$F4_{v2.16}$			MXL2		
	D	Mem	$Time$	D	Mem	$Time$	D	Mem	$Time$
79	3	488	34	4	301	26	3	224	31
89	3	841	82	4	1519	153	3	285	39
99	3	1360	164	4	2883	320	3	454	71
109	3	2093	328	4	2039	241	3	674	117
119	3	3103	622	4	6873	972	3	1473	347
129	3	4475	1194	4	11542	1899	3	1573	312
139	3	6277	2113	4	8701	1238	3	2253	451
149	3	8606	3686	4	24105	5397	3	3151	659
159	3	11546	6645	4	30177	6734	3	4318	960
169	3	15195	10451	4	47787	14800	3	5812	1449
179	3	19741	15801	4	46064	14122	3	7698	1907
189	3	25262	25386	4	91720	41805	3	14154	3782
199	3	31852	40618	4	134144	124278	3	12944	3633
209	3	39813	64753	4	97898	69144	3	16472	6730
219	3	49129	85635	Ran out of memory			3	20736	8165
229	3	60231	1.83D	Ran out of memory			3	36617	4.13H
239	3	73006	2.36D	Ran out of memory			3	31922	3.62H
249	3	87,908	3.42D	Ran out of memory			3	39,098	4.34H
259	3	105,012	4.15D	Ran out of memory			3	47,512	6.51H
...	.	.....	.....	.	.....	.....	.	.....	.....
299	Ran out of memory			Ran out of memory			3	95,317	11.28H

Table 4.11: MXL2: Results for Poly-Dragon-259Var

Round	eD	Matrix	Rank	#Mutants	#LM	#Uni
1	2	259×33671	259	0	0	0
2	3	517×2862727	517	0	0	0
3	3	67340×2895880	67339	776	258	2
4	2	1035×33671	792	256	256	138
5	2	1033×33671	895	118	118	0
6	2	14937×33671	7140	1	1	1
7	2	7140×33671	7021	118	118	118

Table 4.12: F4: Results for Poly-Dragon-259Var

Step	sD	Matrix	#Pairs
1	2	259×33671	251
2	3	67349×2895880	4694
3	2	34446×33671	777
4	3	2835604×2832190	20423

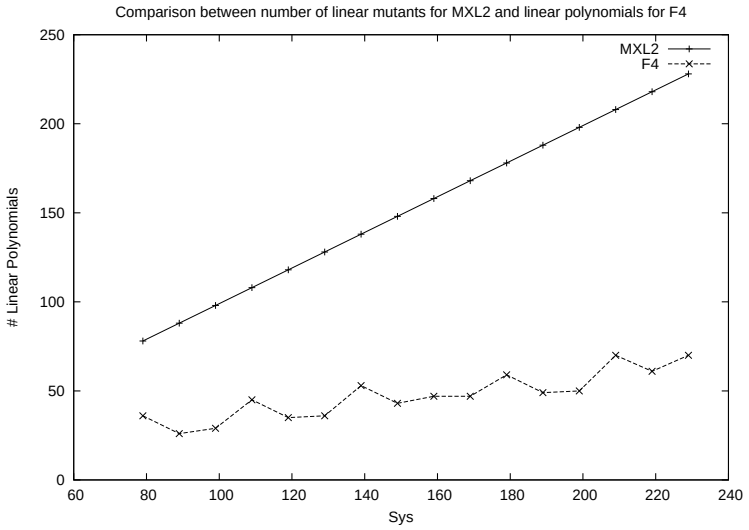

Figure 4.5: LD2:Number of linear Mutants for MXL2 and linear polynomials for F4$_{v2.16}$

4.9 Conclusion

We can conclude that the MutantXL algorithm can indeed outperform the XL algorithm and can solve multivariate systems at a lower degree than the usual XL algorithm. Since the total degree which the XL algorithm needs to go up is typically the bottle neck of this algorithm, this is quite a considerable improvement.

We can also conclude that the MXL2 algorithm is an efficient improvement over the original MutantXL in the case of $\mathbb{F}_2$. Not only can MXL2 solve multivariate systems at a lower degree than the usual XL but also can solve these systems using a smaller number of polynomials than the original MutantXL, since we produce all possible new equations without enlarging the number of the monomials. Therefore, the size of matrices constructed by MXL2 is much smaller than matrices constructed by the original MutantXL. We apply the mutant strategy to two different types of systems, namely random quadratic and HFE. We believe that mutant strategy is a general approach that can improve most of multivariate polynomial solving algorithms.

We present an efficient algebraic cryptanalysis for the Little Dragon Two, LD2, and Poly-Dragon public-key cryptosystems that were claimed to resist algebraic cryptanalysis. In our attack we are able to break LD2 with message length up to 229 bits and Poly-Dragon with message length up to 299 bits using both Magma's F4 and MXL2. In MXL2 algebraic attack, the LD2 and Poly-Dragon schemes are solved at degree three which reflexes the weakness and contradicts the security claims for these two schemes. In all experiments, MXL2 outperforms the used versions of Magma's F4. We realized that the last version of Magma's F4, namely F4$_{v2.16}$, is not so well suitable for solving LD2 and Poly-Dragon systems.

5 Improvements based on the Parallelized Wiedemann Algorithm

In this chapter, a discussion of improving the XL algorithm using the block Wiedemann algorithm as a solver instead of Gaussian elimination is presented. The WXL algorithm over $\mathbb{F}_2$ is introduced. The PWXL algorithm which is a parallel implementation of WXL is discussed. The experiments for using both WXL and PWXL are presented. These experiments are accomplished for dense random systems and HFE systems over $\mathbb{F}_2$. By using PWXL, we are able to solve HFE systems of univariate degree 4352 up to 37 equations in 37 variables in almost 8 days using 81 processors, which was never done before by any known algebraic solver.

5.1 Introduction

The XL algorithm is an algorithm for solving systems of multivariate polynomial equations over finite fields. XL expands the initial system by multiplying it with monomials below a certain degree. XL then linearizes the expanded system in the so-called Macaulay matrix and solves the linear system by Gaussian elimination. See Chapter 3, for more details.

Indeed, the Macaulay matrix of the linearized extended system has rows representing multiples of original polynomials and columns representing monomials up to a given degree. The number of non-zero elements in each row is bounded by the maximum number of monomials that appear in one original polynomial. Therefore, the whole matrix tends to be sparse. In this case, using Gaussian elimination for the linear algebra step increases the density of the matrix due to the fill-in property. Computing the row echelon form to such matrix by F4 and/or XL makes the linear algebra very costly in terms of memory and time. For example, the F4 algorithm that is implemented in Magma was not able to solve a dense random system with 32 quadratic equations in 32 variables on a server that has 128GB RAM.

Suppose that we have a matrix $\mathbf{A}$ over $\mathbb{F}_2$ with dimensions $1,000,000 \times 1,000,000$, in order to use Gaussian elimination to solve a linear system $\mathbf{Ax = b}$, we need at least 116GB of storage to store the entire matrix element-by-element. Even if such a matrix $\mathbf{A}$ were sparse, this amount of storage would not be changed; after several pivoting operations, a sparse matrix would rapidly become dense. It is well-known that Gaussian elimination is an $\mathcal{O}(N^3)$ algorithm for $N \times N$ matrices. In particular, it takes about $\frac{N^3}{3}$ additions and the same amount of multiplications. The matrix $\mathbf{A}$, with $N = 1000000$, would require about 6.6×10^{17} total operations. Therefore, using Gaussian elimination in such a scenario is not reasonable. Variants of Gaussian elimination like structured Gaussian elimination, M4RI algorithm and PLS matrix decomposition perform better than naive Gaussian elimination but the runtime and storage problems still steadfastly exist at even slightly larger matrix sizes. In addition to the high cost of the linear algebra step (Gaussian elimination) in XL, the disability of solving systems with multiple solutions is considered another drawback of XL.

In this context, sparse linear algebra can help to solve these drawbacks. It is realized that the Wiedemann algorithm is one of the well-known algorithms that is applicable to sparse matrices over finite fields. On the other hand, sparse linear algebra was successfully used in integer factorization specially the use of the block Wiedemann algorithm over $\mathbb{F}_2$ in the number field sieve method to factor the RSA-768 number [108]. This leads to the importance of using the Wiedemann algorithm as a solver instead of Gaussian elimination.

In this chapter, we represent an algorithm that uses the block Wiedemann algorithm over $\mathbb{F}_2$ as a solver instead of Gaussian elimination in the XL algorithm, we call it WXL. This idea was used over $\mathbb{F}_{256}$ with a scalar version of the Wiedemann algorithm in [185]. We present an experimental comparison with Magma's implementation of the F4 algorithm, MXL3 [123], an efficient algorithm for computing Gröbner basis, and WXL. Our experiments are based on random instances of the MQ-problem and some HFE cryptosystems that demonstrate the effect of using such solver. The main contribution of this chapter is to show that a parallel version of WXL, we call it PWXL, is able to solve an instance of the HFE systems of univariate degree 4352, the same degree as HFE challenge-2, that are generated over $\mathbb{F}_2$ with 37 quadratic equations in 37 variables using 25.5GB and 81 processors while Magma's F4 and MXL3 can not solve any of such systems with more than 31 variables using 128GB memory.

This chapter is organized as follows: In Section 5.2, an overview of the Wiedemann algorithm is introduced. We then present the WXL algorithm in Section 5.3. In Section 5.4, we introduce experimental results on HFE systems and random systems. A discussion for PWXL is presented in Section 5.5 followed by its experimental results in Section 5.6. Finally we conclude the chapter in Section 5.7.

5.2 The Wiedemann Algorithm

In 1986 Wiedemann introduced an algorithm to solve a linear system and compute the determinant of a black box matrix over a finite field [174]. Wiedemann's approach uses the fact that the minimal polynomial of a matrix generates the Krylov sequences of the matrix and their projections. In other words, when a square matrix is repeatedly applied to a vector, the resulting sequence is linear recursive. In 1994 Coppersmith introduced a block version of Wiedemann's algorithm over $\mathbb{F}_2$ [35]. By using projections of a block of vectors instead of a single vector it is possible to do the parallelization of the matrix times vector products.

Consider the system **Ax = b** over a finite field $\mathbb{F}_q$, with **A** a non-singular, sparse $N \times N$ matrix. The approach used by Wiedemann is to start from a vector **b** and to compute the Krylov sequence $\{uA^ib\}_{i=0}^{2N-1}$, for any row vector $u \in \mathbb{F}_q^{1\times N}$. This sequence is linearly generated since the set of all theses vectors has dimension $\leq N$, so there exists a non-trivial linear dependency relation between the first $N+1$ vectors of this sequence. Moreover, this sequence provides a minimal recurrence polynomial $F_u^{A,b}(\lambda)$ that can be computed using Berlekamp-Massey algorithm. Wiedemann proved that for random vectors u and b with high probability $F_u^{A,b}(\lambda) = F^A(\lambda)$, where F^A is the minimum polynomial of the matrix **A**.

Let $F_u^{A,b}(\lambda) = c_0 + c_1\lambda + c_2\lambda^2 + ... + c_d\lambda^d, c_0, c_1, .., c_d \in \mathbb{F}_q$, and $c_0 = 1$. The vectors of the Krylov sequence satisfy the linear equation $F^{A,b}(A)b = 0$. Hence $c_0b + c_1Ab + c_2A^2b + ... + c_dA^db = 0$, rearranging we obtain $b = -A(c_1b + c_2Ab + ... + c_dA^{d-1}b)$. If we define $x = -(c_1b + c_2Ab + ... + c_dA^{d-1}b)$ then x is a solution of **Ax = b**. For the case that $c_0 = 0$, **A** is

singular, then we have $c_1Ab + c_2A^2b + ... + c_dA^{d-1}b = 0$ or $A(c_1b + c_2Ab + ... + c_dA^{d-1}b) = 0$. So, we can either find a solution of $\mathbf{Ax = b}$ or a non-trivial element in the kernel, $\ker(A)$, of A.

The block Wiedemann algorithm over $\mathbb{F}_2$, for example, uses two block projections $X \in \mathbb{F}_2^{m,N}$ and $Z \in \mathbb{F}_2^{N,n}$, where m and n are two chosen integers, to construct a block matrix sequence $\{X^T A^i Y\}_{i=0}^{\frac{N}{m}+\frac{N}{n}+\mathcal{O}(1)}$, where $Y = AZ$, instead of the scalar sequence of Wiedemann algorithm. This matrix sequence is linearly generated by not only a scalar polynomial, but also by vector and matrix polynomials. It also has both a minimal generating polynomial and a minimal generating matrix polynomial [168].

The computation of the minimal generating matrix polynomial of the block Wiedemann sequence is an important task. Several algorithms have been introduced to deal with this task. Coppersmith uses a multivariate generalization of the Berlekamp-Massey algorithm, Kaltofen solves a homogeneous block Toeplitz system and Villard proposes the using of Fast Power Hermite-Padé solver (FPHPS) algorithm of Backermann and Labahn [168]. While Emmanuel Thomé [165] presents an algorithm, adapted from the divide-and-conquer approach that yielded the HGCD (half-gcd) algorithm and the PRSDC (polynomial reminder sequence by divide-and-conquer) algorithm.

The block Wiedemann algorithm over $\mathbb{F}_2$ is presented in Algorithm 5.1. It depends on the two parameters m, $n \in \mathbb{N}$. The input is a matrix $\mathbf{A}$ of $N \times N$ over $\mathbb{F}_2$. The output is n solutions of $\mathbf{A}w = 0$. Block Wiedemann is consisted of five steps. In the first step, Line 6, two random matrices $X \in \mathbb{F}_2^{m,N}$ and $Z \in \mathbb{F}_2^{N,n}$ are chosen and $Y = AZ$ is computed. Practically, X is chosen as a unit matrix while Z is chosen randomly. The second step, Line 7, is responsible for computing the scalar products $a_i = X^T A^i Y$, for $i = 1, \ldots, \frac{N}{m} + \frac{N}{n} + \mathcal{O}(1)$. In the third step, Line 8, the generating matrix polynomial of the sequence a_i is computed. This generator is an $n \times n$ matrix $F(X)$ of polynomials of degree $\frac{N}{n}$ such that

$$w = \sum_{i=0}^{\frac{N}{n}} A^i Y F_i^T \in Ker(A).$$

The computation of w is the main task of the fourth step, Line 9. With high probability $\mathbf{A}w = 0$ holds. The vectors w for which this hold are printed as solutions in the fifth step, Line 10-11.

The choice of the two parameters m and n is very important in terms of complexity. On the one hand, the bigger m and n, the shorter the computation of the scalar products a_i's. On the other hand, the bigger m and n, the more tedious the computation of a solution particularly the linear generator step. A discussion of the complexity analysis of each step can be found in [6]. The analysis of the efficiency and reliability of the block Wiedemann for computations over large fields by Kaltofen and over small fields by Villard can be found in [102] and [169], respectively.

5.3 WXL: The Wiedemann XL Algorithm

As XL, the WXL algorithm starts with linearizing the original system of polynomial equations to construct a Macaulay matrix of the system at starting degree $D = 2$. This is achieved by replacing each monomial by a new variable. If the constructed Macaulay matrix is not undetermined then we can apply the Wiedemann algorithm to try to solve, otherwise like XL extends the system to the next higher degree $D + 1$ and we repeat the linearization. In the case that we found a

Algorithm 5.1 bWiedemann

Inputs
  $A \in \mathbb{F}_2^{N \times N}$
Output
  $w \neq 0$ such that $Aw = 0$.
Begin
Pick up random matrices $X \in \mathbb{F}_2^{m,N}$, $Z \in \mathbb{F}_2^{N,n}$. Let $Y = AZ$
Let $\delta_l = \lceil N/m \rceil$ and $\delta_r = \lfloor N/n \rfloor$. Compute $a_i = XA^iY, i = 0, \ldots, \delta_l + \delta_r - 1$.
Compute a generating vector polynomial $g(\lambda) = c_l\lambda^l + c_{l+1}\lambda^{l+1} + \cdots + c_d\lambda^d \in \mathbb{F}_2^n[\lambda]$ of degree at most δ_r for the sequence $\{XA^iY\}$, where $l \geq 0$, $d \leq \delta_r$, $c_l \neq 0$.
Compute $\hat{w} \leftarrow c_lZ + c_{l+1}AZ + \cdots + c_dA^{d-l}Z$;
(with high probability $\hat{w} \neq 0$ and $A^{l+1}\hat{w} = 0$)
Compute the first k such that $A^k\hat{w} = 0$;
If $k \geq 1$ then $w = A^{k-1}\hat{w}$ else $w = 0$
End

determined or overdetermined matrix then we try to solve it by extracting a square sub-matrix from the extended system at that degree. If there is a solution, we must check whether such a solution for the linearized system is also a solution to the original quadratic system or not. If this solution is satisfiable to the original system then terminate and return the solution, otherwise we may try some other square sub-matrix to be solved again until some limit. After trying up to some limit and there is no solution then extend the system and linearize again. Algorithm 5.2 describes the WXL algorithm.

The main critical point of WXL is to choose a number of polynomials in order to construct a square Macaulay matrix of the linearized system at a certain degree that can generate as small number of solution as possible. We use a heuristic argument, by Ding, from [185] that if we pick rows at random under the constraint that we have enough equations at each degree, then usually we have a linearly independent set. This is exactly what we mean by the function "Make_square($\mathcal{M}^{acaulay}$)" in the WXL algorithm. In all experiments of HFE and dense random systems, WXL always solves using only the first square sub-matrix at a certain degree D while for some very sparse random systems, it needs to select more than one such square sub-matrix.

In the WXL algorithm, by "Extend(P, D)" we mean, multiply each polynomial by all monomials of degree $D-2$, see Algorithm 5.3. "Wiedemann($\mathcal{M}_{sq}^{acaulay}$)" applies the block Wiedemann algorithm to a square Macaulay matrix as it is described in Algorithm 5.1. The "Check_solution(P, $Wsolution$)" procedure is responsible for substituting with the solutions generated by the Wiedemann algorithm into the original quadratic system and checks whether these solutions are satisfied or not.

5.4 WXL Experimental Results

In this section we present experimental results and compare the performance of WXL with Magma's implementation of F4 and MXL3. We are interested in solving systems of multivariate quadratic polynomial equations when the number of equations is the same as the number of unknowns. We use some instances of dense random systems generated by Courtois [43], denoted by MQR followed by the number of variables and some HFE systems generated by the code of John Baena, denoted by HFEbae(d, m, n) where d is the degree of HFE univariate polynomial, m is the num-

Algorithm 5.2 WXL

Inputs
  P: set of m quadratic polynomials.
  $Limit$: number of maximum trails.
Output
  $Solution$: A solution of P=0.
Variables
  $\mathcal{M}^{acaulay}$: a matrix whose entries are the coefficients of a system of multivariate polynomial equations in graded lex order.
  $\mathcal{M}_{sq}^{acaulay}$: a square submatrix.
  $\tilde{P}$: set of all polynomials that are included in the system.
  D: the highest degree of $\tilde{P}$.
  $solved$: a flag to indicate whether the system is solved or not.
  $attempt$: a counter for the number of trails.
  $Wsolution$: set of solutions generated by Wiedemann for the linearized system.
Begin
Initialization()
$\{\mathcal{M}^{acaulay}, Solution, Wsolution \leftarrow \varnothing, solved \leftarrow False, \tilde{P} \leftarrow P, D \leftarrow 2\ \}$
repeat
  $\mathcal{M}^{acaulay} \leftarrow$ Linearize($\tilde{P}$)
  if $nRows(\mathcal{M}^{acaulay}) \geq nCols(\mathcal{M}^{acaulay})$ **then**
    $attempt \leftarrow 1$
    repeat
      $\mathcal{M}_{sq}^{acaulay} \leftarrow$ Make_square($\mathcal{M}^{acaulay}$)
      $Wsolution \leftarrow$ Wiedemann($\mathcal{M}_{sq}^{acaulay}$)
      if $Wsolution \neq \varnothing$ **then**
        ($solved$,$Solution$) $\leftarrow$ Check_solution(P, $Wsolution$)
        if $solved$ **then**
          Return ($Solution$)
        end if
      end if
      $attempt \leftarrow attempt + 1$
    until ($attempt \geq Limit$)
  end if
  $D \leftarrow D + 1$
  $\tilde{P} \leftarrow \tilde{P} \cup$ Extend(P, D)
until ($solved$)
End

Algorithm 5.3 Extend(P, D)

Inputs
  P: set of m quadratic polynomials.
  D: the highest degree of P.
Output
  $new\_polynomials$: a set of multiplied polynomials by monomials.
Variables
  $Monomial_D$: The set of all monomials of degree D.
Begin
  $new\_polynomials \leftarrow \varnothing$
  $Monomial_{D-2} \leftarrow$ all monomials of degree $D-2$
for all $p \in P$ **do**
  for all $m \in Monomial_{D-2}$ **do**
    $new\_polynomials \leftarrow new\_polynomials \cup \{m \times p\}$
  end for
end for
D $\leftarrow$ D+1.
End

ber of equations and n is the number of variables. All the dense random systems have multiple solutions except for the instance with 24 variables has a unique solution. The central map for the HFE scheme is not necessarily a bijection, therefore we may find more than one solution to such systems.

The complexity of solving systems of dense multivariate quadratic polynomial equations depends on the number of variables and the number of equations in each system. Therefore, the complexity of solving different systems with the same number of variables and equations more or less will be the same. In this framework, the results given in this section are for an instance for each system. All the experiments are done on a Sun X4440 server, with four "Quad-Core AMD Opteron^{TM}Processor 8356" CPUs and 128GB of main memory. Each CPU is running at 2.3 GHz. In these experiments we use only one out of the 16 cores.

We used Magma version (V2.16) unless stated. The WXL algorithm we implemented using the block Wiedemann solver written by Thorsten Kleinjung which uses 64 bit word block Wiedemann and returns 64 solutions. It also uses MSLGDC (Matrix Sequences Linear Generator by Divide-and Conquer) algorithm for computing linear generator in subquadratic computation [165]. All experimental data for MXL3 are done by Mohamed Saied Emam Mohamed, the first author of [123].

Tables 5.1 and 5.2 show results for HFE systems of univariate degree 4352 that are generated directly over $\mathbb{F}_2$ and results for dense random systems, respectively. The first column "Sys" denotes the number of variables and the number of equations for each system. The highest degree of the elements of the system can reach is denoted by "D". The used memory in Megabytes (MiB) or Gigabytes (GB) and the execution time in seconds (S), minutes (M), hours (H) or days (D) are represented by "Mem" and "Time", respectively.

In both tables, we can see that WXL always outperforms F4 in terms of memory. Therefore, Magma's F4 version (V2.16) is faster than WXL. Magma's F4 is using a fast, improved, and

Table 5.1: Performance of WXL compared to F4 and MXL3 for HFEbae(4352, n, n)

n	$F4_{v2.16}$			MXL3			WXL		
	D	Mem	Time	D	Mem	Time	D	Mem	Time
24	6	3.5GB	5M	6	390MiB	6M	6	436MiB	13M
25	6	7.4GB	5M	6	607MiB	13M	6	524MiB	22M
26	6	11.6GB	10M	6	1.2GB	24M	6	814MiB	39M
27	6	17.3GB	22M	6	2.4GB	1.1H	6	920MiB	1.1H
28	6	21.6GB	50M	6	4.7GB	3.0H	6	1.0GB	2.0H
29	6	31.6GB	1.9H	6	9.4GB	5.5H	6	1.2GB	3.3H
30	6	92.9GB	17.0H	6	14.7GB	9.4H	7	9.0GB	4.6D
31	6	113.2GB	1.1D	6	22.6GB	1.1D	7	19.4GB	5.1D

Table 5.2: Performance of WXL compared to F4 and MXL3 for dense random systems

Sys	$F4_{v2.16}$			MXL3			WXL		
	D	Mem	Time	D	Mem	Time	D	Mem	Time
MQR24	6	3.5GB	4M	6	392MiB	6M	6	514MiB	13M
MQR25	6	7.4GB	8M	6	698MiB	12M	6	578MiB	22M
MQR26	6	11.6GB	16M	6	1.2GB	24M	6	749MiB	40M
MQR27	6	17.3GB	37M	6	2.3GB	48M	6	1.0GB	1.3H
MQR28	6	21.6GB	1.1H	6	4.7GB	2.2H	6	1.4GB	2.0H
MQR29	6	31.6GB	2.1H	6	9.2GB	5.2H	6	1.8GB	3.1H

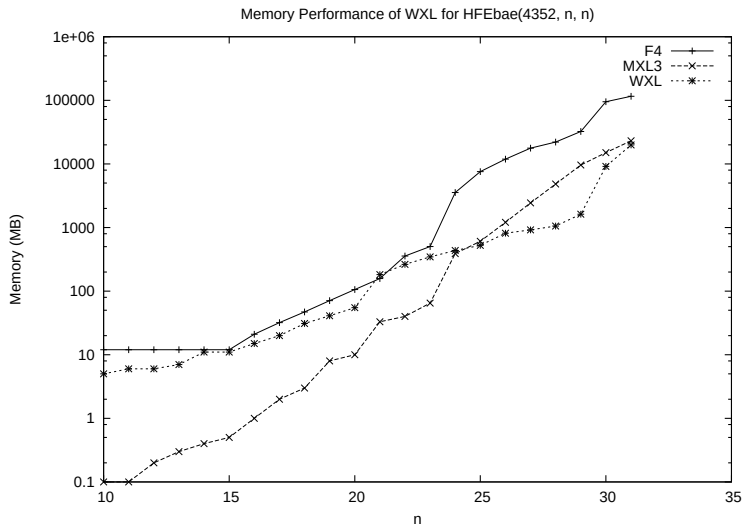

Figure 5.1: Performance of WXL compared to F4 and MXL3 for HFE(4352, 10-31, 10-31)

Table 5.3: Performance of WXL versus F4 for dense random systems

Sys	$F4_{v2.13-10}$			WXL		
	D	Mem	Time	D	Mem	Time
MQR24	6	3.0GB	14M	6	514MiB	13M
MQR25	6	5.0GB	22M	6	578MiB	22M
MQR26	6	8.2GB	55M	6	749MiB	40M
MQR27	6	13.0GB	1.8H	6	1.0GB	1.3H
MQR28	6	20.0GB	3.8H	6	1.4GB	2.0H
MQR29	6	29.3GB	7.1H	6	1.8GB	3.1H

Table 5.4: Matrix dimensions for MXL3, F4, and WXL for dense random systems

Sys	MXL3	$F4_{v2.13-10}$	WXL
MQR24	57183×57171	207150×78637	190051×190051
MQR25	66631×76414	248795×109046	245506×245506
MQR26	88513×102246	298673×148885	313912×313912
MQR27	123938×140344	354294×198007	397594×397594
MQR28	201636×197051	420773×261160	499178×499178
MQR29	279288×281192	499258×340290	621616×621616

updated linear algebra. For older versions, Table 5.3 shows that WXL is faster and uses less memory than F4 version (V2.13-10).

WXL also outperforms MXL3 in terms of memory for systems with number of variables greater than 24 and for systems with 28-29 variables, WXL is faster and consumes less memory than MXL3. Starting from 30 variables, WXL is worse in terms of time against MXL3, this is because WXL solves at degree 7 while MXL3 solves at degree 6. Figure 5.1 shows a comparison compared to F4, MXL3, and WXL in terms of memory for HFE systems that have a number of variables 10-31.

In Table 5.4, we compare the matrix dimensions with MXL3, F4, and WXL. It is obvious that WXL has the biggest matrix dimensions because WXL did not have an optimized selection strategy for extending polynomials which is the key for MXL3 and F4.

We realized that MXL3 is better than WXL in both memory and time for systems that have number of variables less than 25. The main reason for that is that MXL3 has an optimized selection strategy that makes the systems be solved with a very small matrix dimensions compared to WXL. While WXL is better only in memory but not in time for systems that have number of variables greater than 24. For systems that have a number of variables 28 and 29, WXL is better in both time and memory. MXL3 uses mutants that make some systems be solved at lower degree than WXL. This is the reason that WXL takes a lot of time in the instances that have 30 and 31 variables.

5.5 PWXL: The Parallel Wiedemann XL Algorithm

In this section we discuss the parallelized version of the WXL algorithm. Obviously, the last version of Magma's implementation of F4 is faster than WXL. One way to improve WXL is to use more than one processor based on the advantage that the Wiedemann algorithm is applicable to be parallelized while F4 is not easy to be parallelized.

In the context that the block Wiedemann algorithm is accomplished using five different steps, see Section 5.2, the bottlenecks of this algorithm are step 2, Line 7 Algorithm 5.1, and step 4, Line 9 Algorithm 5.1. In the second step, the computation of the scalar products is performed. The fourth step is responsible for the evaluation of the solution based on the matrix polynomial computed in the third step. The basic operation behind these two steps is the matrix-by-vector product. Indeed, this operation can be parallelized in some different ways to gain better performance.

The first method to parallelize the matrix-by-vector multiplication is to copy the whole matrix $\mathbf{A}$ and X^T to every node and distribute block of vectors from Y. The key idea here is that the k^{th} block column $X^T AY_k$ of the sequence $X^T AY$ is computed independently from other blocks. Therefore, each node could multiply independently the block vector Y_k from Y by the matrix A. Afterwards, the resulting block vector AY_k is multiplied by the matrix X^T. As an outcome, we obtain the block $X^T AY_k$ based on the block Y_k. This method can be illustrated by Figure 5.2. The second method is to copy both X^T and Y to every node and partition the matrix $\mathbf{A}$ as row blocks, see Figure 5.3. Another way is to copy X^T to each node and partition the matrix $\mathbf{A}$ as column partitions as well as Y is partitioned in vertical blocks, see Figure 5.4. The main two advantages for the first method are avoiding synchronization and minimizing the cost of data communication across a network. Therefore, it is the preferred method by [103] while in [108], the authors used a mixed strategy for the second and the third methods; the matrix is split vertically and horizontally.

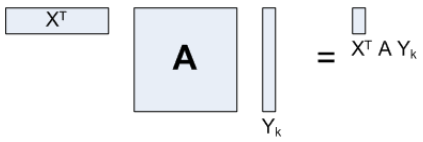

Figure 5.2: Method 1: Parallelize Matrix-by-Vector Multiplication

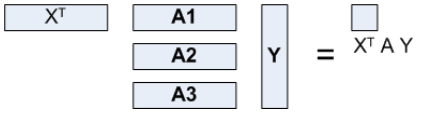

Figure 5.3: Method 2: Parallelize Matrix-by-Vector Multiplication

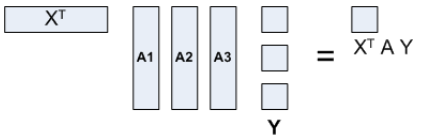

Figure 5.4: Method 3: Parallelize Matrix-by-Vector Multiplication

There are several implementations of the block Wiedemann algorithm. We test WLSS2 [103] and Kleinjung's code that is used to factor RSA-768 number [108]. The latter is more efficient than WLSS2. Therefore, we focus only on it to be the solver for our PWXL algorithm.

Kleinjung's code is implemented in C and parallelized by MPI (Message Passing Interface). It consists of five separated programs that communicate by means of files. Each program is corresponding to a step in Algorithm 5.1. We add to these five programs another two. The first one is for preprocessing that is responsible for extending the original system up to a certain degree, select randomly a square matrix from the extended system, then convert the square matrix to the proper format which is accepted by the first program in Kleinjung's code. The second program is responsible for checking the solution that is generated from the Wiedemann algorithm and returns it such that it is also a solution for the original system otherwise it returns a no solution message.

The run time of the Wiedemann algorithm depends on the number P of processors, but in a more complex way. Basically, there are local computations and there is a communication between processors. The later depends on the topology of the network; a torus topology of $P = P_1 \times P_2$ processors with $P_1 \approx P_2$ seems to be a good choice.

A future big advantage of PWXL is the fact that the linear systems can be generated in a distributed fashion. Since we know what the initial system is and what the effect of the multiplication by monomials is, we can generate the parts of the system on the respective nodes without the need to store the full linear system in one place at any time.

5.6 PWXL Experimental Results

In this section, we present the promising results that can be obtained from the PWXL algorithm. The experimental server for these experiments is a SUN X4440, with four "Six-Core AMD Opteron^{TM}Processor 8435" CPUs running at 2.6 GHz each, 24 Cores in total and 64 GB System memory which is called userver4 in CDC, Informatik, TUD.

In Table 5.5, we compare the performance of the PWXL algorithm in memory and time using 1, 2, 4, 8, 16 processors to HFE systems with univariate degree 4352 for 24-36 equations in 24-36 variables. The used memory in Gigabytes (GB) and the execution time in hours (H) or days (D) is represented by "Mem" and "Time", respectively. The disk space measured in Migabytes (MiB) or Gigabytes (GB) which is used by PWXL is represented in the "DS" column.

In [123], the authors noticed that when MXL3 and F4 tried to solve a 32 variable system, both solvers were unable to extend the system to degree 7 because of memory. While PWXL solves systems starting from number of variables equal to 30 at degree 7. Also, PWXL can successfully solve an instance of HFE(4352, 36, 36) in 28.3 days using 16 processors.

Table 5.6 is the same as Table 5.5 except that we use 16, 25, 36, 49, 64 and 81 processors in HHLR-GU Clusters at Center for Scientific Computing (CSC) of the Goethe University, Frankfurt am Main. Using HHLR-GU, we are able to solve an HFE instance of univariate degree 4352 with 37 variables and equations in 7.5 days using 25.5GB RAM. In this table, the used memory and the execution time are for the block Wiedemann algorithm step.

Table 5.5: PWXL Performance for HFEbae(4352, n, n) Systems using userver4

n	DS	1P		2P		4P		8P		16P	
		Mem	Time	Mem	Time	Mem	Time	Mem	Time	Mem	Time
24	71MiB	93MiB	12M	120MiB	7M	142MiB	4M	157MiB	2M	170MiB	2M
25	106MiB	177MiB	22M	254MiB	13M	374MiB	7M	485MiB	5M	969MiB	3M
26	154MiB	231MiB	40M	241MiB	22M	389MiB	12M	527MiB	9M	612MiB	4M
27	231MiB	291MiB	1.2H	426MiB	40M	535MiB	23M	596MIB	13M	771MiB	8M
28	320MiB	376MiB	2.1H	537MiB	1.1H	684MiB	48M	850MiB	27M	944MiB	15M
29	463MiB	566MiB	3.8H	670MiB	2.2H	774MiB	1.2H	960MiB	45M	1.1GB	32M
30	3.9GB	2.6GB	4.62D	2.7GB	3.26D	2.8GB	2.45D	3.4GB	22.3H	3.7GB	13.22H
31	6.2GB	2.6GB	12.33D	2.7GB	6.41D	2.9GB	4.07D	3.5GB	2.88D	3.9GB	23.93H
32	9.4GB	3.5GB	16.70D	3.6GB	11.72D	3.9GB	6.13D	4.6GB	4.93D	4.7GB	1.35D
33	14.4GB	4.7GB	32.79D	4.9GB	20.27D	5.2GB	11.46D	5.6GB	7.93D	6.3GB	3.2D
34	21.7GB									9.4GB	8.03D
35	32.5GB									10.9GB	12.45D
36	48.1GB									14.1GB	28.23D

Table 5.6: PWXL Performance for HFEbae(4352, n, n) Systems using HHLR

n	DS	16P		25P		36P		49P		64P		81P	
		Mem	Time	Mem	Time	Mem	Time	Mem	Time	Mem	Time	Mem	Time
30	3.9GB	2.34GB	9.37H	2.97GB	7.0H	3.51GB	7.3H	4.7GB	5.1H	6.4GB	5.0H	8.47GB	4.5H
31	6.2GB	3.0GB	18.1H	3.68GB	14.1H	5.22GB	11.9H	7.13GB	7.9H	9.61GB	7.27H	12.4GB	6.8H
32	9.4GB	4.02GB	1.45D	4.56GB	1.04D	5.54GB	18.7H	7.5GB	13.4H	10.1GB	11.2H	13.0GB	11.6H
33	14.4GB	5.1GB	2.66D	5.9GB	1.96D	6.7GB	1.92D	7.9GB	1.16D	9.6GB	20.4H	15.5GB	18.8H
34	21.7GB	6.7GB	4.28D	7.5GB	2.89D	8.6GB	2.7D	11.4GB	1.51D	15.0GB	1.58D	19.1GB	1.9D
35	32.5GB	8.6GB	5.3D	9.7GB	5.03D	10.9GB	4.19D	13.1GB	2.73D	17.1GB	2.58D	21.9GB	2.36D
36	48.1GB	11.5GB	11.6D	12.7GB	8.23D	14.2GB	5.84D	16.9GB	5.15D	21.4GB	4.4D	27.3GB	3.59D
37	70.0GB											25.5GB	7.5D

5.7 Conclusion

In this chapter, we present the WXL algorithm for solving systems of multivariate quadratic polynomial equations over $\mathbb{F}_2$ that is based on the block Wiedemann algorithm instead of Gaussian elimination as a solver. Experimentally, WXL is better than Magma's F4, which is known to be the best available implementation of F4, in terms of memory for HFE and dense random systems. Moreover, by using PWXL, a parallelized version of WXL, we are able to solve systems with higher number of variables that aren't solved by other algebraic solvers.

We are interested in solving instances of random and HFE systems of univariate degree 4352 which is the same degree of the HFE challenge-2. From experimental point of view, we can conclude that HFE systems of univariate degree 4352 over $\mathbb{F}_2$ have the same complexity of random systems within the number of variables from 10 to 37.

We plan to use more processors in order to push PWXL to solve systems with more number of variables taking into account the number of processors versus the number of variables. We also intend to use some ideas from structured Gaussian elimination to minimize the matrix dimensions and at the same time keep the sparsity not changed as possible.

The PWXL implementation is applicable only for systems that can be solved at a degree where the number of polynomials is greater than or equal to the number of monomials. So we intend to use the concept of mutant to solve such shortage. The mixing between MutantXL and PWXL is a promising tool that can improve the field of algebraic cryptanalysis.

6 Combination of Mutants and using the Wiedemann Solver

This chapter is to discuss a combination of the two suggested improvements for the XL algorithm, mutants strategy and parallelized Wiedemann algorithm. In this chapter, a discussion of how to generate mutants using the block Wiedemann algorithm is presented. How to use these mutants either in MutantXL or in the WMXL, Wiedemann Mutant XL, algorithm is also discussed. Experimental results for different scenarios are introduced.

6.1 Introduction

In symbolic computation, the F4 algorithm is the best known algorithm for solving systems of multivariate polynomial equations. In cryptography, the XL algorithm was introduced as a tool for solving such systems. The overall performance of XL and F4 depend on the degree at which a solution could be found. Practically, the running time and memory consumption for XL is greater than in the case of the F4 algorithm.

One way to fill in the gap between XL and F4 is to use the concept of mutant. Based on that concept, we introduced the MXL2 algorithm that is an improvement of MutantXL which in turn is an improvement of the XL algorithm, see Chapter 4 for more details. Another way to fill in this gap is to use a parallelized sparse linear algebra solver such as block Wiedemann algorithm instead of Gaussian elimination, see Chapter 5 for more details. Another improvement which is not included in the context of this thesis is related to the termination condition of XL. It was shown that XL terminates successfully in the case of unique solution. Improving XL as well as MXL2 to be able to solve systems with multiple solutions, in particular computing the Gröbner basis, was presented in [123].

The main advantage of the mutant-based improvements is that the degree at which a solution could be found is less than or equal to the case of the XL algorithm. The main disadvantage is the dense matrix representation. In this representation, a lot of zeros are represented in memory which waste a huge amount of that memory and the largest matrix that could be represented in memory has a limited size, for example, the largest square matrix dimension using 128GB RAM is 1,048,576. By using a parallelized sparse linear algebra, we could solve the disadvantage of the mutant-based improvements while the degree at which a solution could be found is greater than or equal to the case of the XL algorithm. Therefore, the combination of the main two advantages of the mutant-based improvements and the improvements based on the parallelized Wiedemann algorithm could be a third direction for improving the XL algorithm.

In this chapter, we present an algorithm to generate mutants using the block Wiedemann algorithm over $\mathbb{F}_2$. This idea was pointed out in [98] by generating only linear polynomials. The use of these mutants either in MutantXL or in the WMXL algorithm is introduced. The WMXL is a combination of XL algorithm, Wiedemann algorithm and the mutant strategy. The WMXL and WXL,

see Chapter 5, algorithms are similar in using the block Wiedemann algorithm as a solver. WMXL and MXL2 as well as MutantXL are similar in using the concept of mutants. Like WXL, we use block Wiedemann solver written by Thorsten Kleinjung in implementing WMXL. A variant of WMXL called KXL (Kernel-based XL) is also introduced. This variant is used to generate linear mutants instead of solving in the last step of WMXL. Afterwards, we use MutantXL or MXL2 to solve the quadratic and linear polynomials. We present also experimental comparisons with Magma's implementation of the F4 algorithm, WXL, MutantXL and MXL2. Our experiments are based on random instances of the MQ-problem and some HFE cryptosystems that demonstrate the effect of using WMXL in different scenarios.

This chapter is organized as follows: In Section 6.2, we discuss how to generate mutants using the Wiedemann algorithm. Section 6.3 presents the Wiedemann Mutant XL algorithm and discusses the KXL method. In Section 6.4, we introduce experimental results on HFE systems and random systems in some specific cases. The conclusion of the chapter is presented in Section 6.5.

6.2 Generating Mutants using the Block Wiedemann Algorithm

In Chapter 4, the concept of mutant was defined as follows: Let $I = \langle p_1, \ldots, p_m \rangle$ be the ideal generated by the set $P = \{p_1, \ldots, p_m\}$ where $p_1, \ldots, p_m$ are multivariate polynomials. Let $\mu \in I$, then μ can be represented as $\mu = f_1 p_1 + \cdots + f_m p_m$. The number $L = max\{deg(f_i p_i) : 1 \leq i \leq m, f_i \neq 0\}$ is defined to be the level of this representation of μ. The Level of μ with respect to P, denoted by $L(\mu)$, is defined to be the minimal level of any representation. If $deg(\mu) < L(\mu)$ then μ is defined to be a **Mutant**.

Let **B*x*** **= b** be the linearized system that is to be solved where **B** is the Macaulay matrix at some degree D. Each line in the matrix **B** describes a polynomial and each column represents the coefficient of a particular monomial in the polynomials. The columns of **B** represent monomials of degree $\leq D$ ordered in a graded lexicographic ordering (grlex) from left to right. Let $\mathbf{A} = [\mathbf{B}|\mathbf{b}]$ be the augmented matrix that is obtained by appending the columns of the matrix **B** and the vector column **b**. Therefore, **A*x*** **= 0** is the homogenization of **B*x*** **= b**.

The matrix **A** can be divided vertically, $[\mathbf{A}_1|\mathbf{A}_2]$, where $\mathbf{A}_1$ is a submatrix with the number of columns equal to the monomials of degree D (i.e. $\mathbf{A}_1$ is a submatrix of **A** consisting of only the coefficients of the highest degree monomials of the polynomials represented by **A**) and $\mathbf{A}_2$ is a submatrix with the number of columns equal to the monomials of degree $< D$. Suppose that ν is a non trivial left kernel vector of $\mathbf{A}_1$. By multiplying ν by the matrix **A**, the highest degree monomials in the polynomials represented by **A** are canceled out. Therefore, $\nu\, \mathbf{A}_2$ represents a mutant. It is also possible to represent the polynomials of **A** as columns, $(\mathbf{A}^T)$, and splitting horizontally. In this case, if ν is a non trivial right kernel vector of $\mathbf{A}_1^T$ then $(\mathbf{A}_2^T \nu)^T$ is a mutant. In order to find the kernel of a matrix $\mathbf{A}_1$, we need to solve the system $\mathbf{A}_1 Y = 0$. One usually does this by putting $\mathbf{A}_1$ in RREF since the matrix $\mathbf{A}_1$ and its Reduced Row Echelon Form (RREF) have the same kernel. Another way is to use Wiedemann algorithm to solve $\mathbf{A}_1 Y = 0$. Therefore, mutants can be generated by using the matrix kernel concept which also can be computed by the Wiedemann algorithm. Generating mutants using Wiedemann algorithm is described in Algorithm 6.1. Generating linear polynomials by building and reducing Macaulay matrices for various degrees was pointed out in [98]. Algorithm 6.1 can be modified to generate only linear polynomials by

using the Algorithm 6.2 which was called Linear method in [98].

The implementation of Algorithm 6.1 is based on the block Wiedemann solver which used in [108]. This solver uses 64 bit word blocks. Therefore, using only one block word will return 64 solutions. The number of solutions obtained from the block Wiedemann algorithm is the number of generated lower degree polynomials. We observed that for some specific systems, all the 64 generated lower degree polynomials are linearly independent. Therefore, these polynomials are mutants. In the context of using such algorithm to generate mutants, we need to generate more than 64 mutants for systems with large number of variables. One way to return more than 64 solutions is to increase the number of block bit words. The total number of solutions is $64 \times BBW$ where BBW is the number of Block Bit Words. The second way is to use a smaller BBW and run the block Wiedemann algorithm several times. Practically, we use BBW up to 50 to generate a total 3200 mutants at a time. In the case that we need more than 3200 mutants, we use the block Wiedemann algorithm several times.

Algorithm 6.1 Generate Mutants

Inputs
  $\mathcal{M}^{acaulay}$: coefficient matrix for all polynomials in the current system.
  D: the highest degree of a set of multivariate polynomial equations.
  n: number of variables.
Output
  Mutants: a set of degree $D-1$ polynomials.
Variables
  $\mathcal{M}_{D}^{acaulay}$: coefficient matrix with degree D polynomials for rows, degree D monomials for columns.
  $\mathcal{M}_{<D}^{acaulay}$: coefficient matrix with degree D polynomials for rows, degree $< D$ monomials for columns.
  $\mathcal{M}_{sq}^{acaulay}$: a square submatrix.
  Ker: a set of kernel elements.
Begin
Initialization()
$\mathcal{M}_{sq}^{acaulay} \leftarrow$ Make_square($\mathcal{M}^{acaulay}$)
($\mathcal{M}_{D}^{acaulay}$, $\mathcal{M}_{<D}^{acaulay}$) $\leftarrow$ Split_matrix($\mathcal{M}_{sq}^{acaulay}$, D, n)
$Ker \leftarrow$ Wiedemann($\mathcal{M}_{D}^{acaulay}$) {Left kernel: $Ker.\mathcal{M}_{D}^{acaulay}$=0 }
if $Ker \neq \{0\}$ **then**
  $Mutants \leftarrow Ker \times \mathcal{M}_{<D}^{acaulay}$
end if
Return $(Mutants)$
End

6.3 WMXL: The Wiedemann Mutant XL Algorithm

The main idea for the WMXL algorithm is to combine XL algorithm, Wiedemann algorithm and mutant strategy in order to obtain a solution for systems of multivariate polynomial equations in

Algorithm 6.2 Generate Linear Mutants

Inputs
  $\mathcal{M}^{acaulay}$: coefficient matrix for all polynomials in the current system.
  D: the highest degree of a set of multivariate polynomial equations.
  n: number of variables.
Output
  $Linear$: a set of linear polynomials.
Variables
  $\mathcal{M}_{>1}^{acaulay}$: coefficient matrix with degree D polynomials for rows, degree > 1 monomials for columns.
  $\mathcal{M}_{\leq 1}^{acaulay}$: coefficient matrix with degree D polynomials for rows, degree ≤ 1 monomials for columns.
  $\mathcal{M}_{sq}^{acaulay}$: a square submatrix.
  Ker: a set of kernel elements.
Begin
Initialization()
$\mathcal{M}_{sq}^{acaulay} \leftarrow$ Make_square($\mathcal{M}^{acaulay}$)
($\mathcal{M}_{>1}^{acaulay}$, $\mathcal{M}_{\leq 1}^{acaulay}$) $\leftarrow$ Split_matrix($\mathcal{M}_{sq}^{acaulay}$, D, n)
$Ker \leftarrow$ Wiedemann($\mathcal{M}_{>1}^{acaulay}$) {Left kernel: $Ker.\mathcal{M}_{>1}^{acaulay}$=0 }
if $Ker \neq \varnothing$ **then**
  $Linear \leftarrow Ker \times \mathcal{M}_{\leq 1}^{acaulay}$
end if
Return $(Linears)$
End

an efficient way. The main operations in the WMXL algorithm are **Linearize**, **Wiedemann**, **Make_square**, **Check_solution**, **Generate_mutants** and **Extend**. The only deferences between WMXL and MXL2 are using the Wiedemann algorithm to solve instead of Gaussian eliminating and mutants generatation process is done by using matrix kernel. The following is a short description for each operation.

- **Linearize**:
  Construct a Macaulay matrix of a system of polynomial equations at some degree D. This is achieved by replacing each monomial by a new variable. The monomials are ordered in a graded lexicographic ordering.
- **Wiedemann**:
  apply Wiedemann algorithm to obtain a solution. See Algorithm 5.1.
- **Make_square**:
  Remove randomly some rows or some columns in order to obtain a square sub-matrix.
- **Check_solution**:
  If there is a solution obtained from the Wiedemann algorithm, we must check whether such a solution is also a solution to the original quadratic system.
- **Generate_mutants**:
  If the system is not solved at certain degree D then try to generate mutants using Wiedemann algorithm as described in Section 6.2.
- **Extend**:
  Multiply each polynomial at some degree D by each variable such that the original polynomial is multiplied by each variable only once. The new resulting polynomials are added to the whole system that is obtained at certain degree D.

As WXL, the WMXL algorithm starts with linearizing the original system of polynomial equations to construct a coefficient matrix of the system at starting degree $D = 2$. If the constructed matrix at some degree has number of rows greater than or equal to the number of columns then we can apply the Wiedemann algorithm to try to solve after extracting a square sub-matrix. If there is a solution, we must check whether such a solution for the linearized system is also a solution to the original quadratic system. If this solution is a solution of the original system then terminate and return the solution, otherwise we try some other square sub-matrix and repeat the process until the number of trials is less than or equal some limit. After trying up to some limit and there is no solution or the coefficient matrix is underdetermined then we try to generate mutants. If there exist some mutants then we multiply these mutants by variables such that each original polynomial is multiplied by each variable only once. The new resulting polynomials are added to the whole system that is obtained at certain degree D. In the case that there is no mutants, the system is extended to the next higher degree $D + 1$ and repeat linearization. The WMXL algorithm is described in Algorithm 6.3.

The WXL algorithm can be obtained from WMXL if the lines 33-36 in the WMXL algorithm are skipped. These lines are responsible for generating and extending mutants then adding the new polynomials to the whole system. For some specific systems, it is better to use WMXL to generate

linear mutants instead of solving such systems using Wiedemann algorithm. This process could be done by changing lines 22-29 in Algorithm 6.3 to two different steps. The first step is to generate linear mutants using Algorithm 6.2. The second step is to add the new generated linear mutants to the original quadratic polynomials then solve using MutantXL, MXL2 or F4 at most degree two. We refer to a variation to WMXL that generates linear mutants in the last step instead of solving as KXL (Kernel-based XL) method.

6.4 Experimental Results

In this section we present experimental results and compare the performance of WMXL with WXL, MutantXL, MXL2 and Magma's implementation of F4. We use some instances of dense random systems, denoted by MQR(n, m) where n is the number of variables and m is the number of equations. In this section we use $m = 1.5n$. We use also some HFE systems generated by the code of John Baena, denoted by HFEbae(d, m, n) where d is the degree of HFE univariate polynomial, m is the number of equations and n is the number of variables. The results given in this section are for an instance for each system.

We used Magma version (V2.16) and (V2.15). For both Magma versions, we use the graded lexicographic ordering and the field equations are included. The WMXL algorithm was implemented based on the the same solver that is used by WXL algorithm, see Chapter 5. We have two different implementations for WMXL. The first one uses only one processor and runs in one program. The second one uses multiple processors and five programs, each program represents one step in the block Wiedemann algorithm.

We present three different scenarios for solving systems of multivariate polynomial equations over $\mathbb{F}_2$ that provide an evidence for the efficiency of using WMXL as a tool for solving such systems. In the first scenario, we use HFE systems of univariate degree $d = 16$. These systems are very structured and solved at degree 3. When we apply WXL to solve these systems, the degree at which WXL finds a solution started at degree 4 for number of variables and equations is equal to 10. This degree will be increased when increasing the number of variables and equations. Therefore the efficiency will be decreased for structured systems which are solved at lower degrees compared to random systems with the same number of variables and equations. In order to recover the shortage for WXL to solve such structured systems, we need to use WMXL algorithm. Figure 6.1 presents the memory performance for WMXL versus WXL for HFEbae(16, n, n). Figure 6.2 shows the maximum degree for WMXL versus WXL. All the experiments for HFEbae(16, n, n) using WXL and WMXL are done on a Sun X4440 server, with four "Quad-Core AMD Opteron^{TM}Processor 8356" CPUs and 128GB of main memory. Each CPU is running at 2.3 GHz. In these experiments we use only one out of the 16 cores. We generate a maximum 64 mutants at a time for WMXL.

In the second scenario, we use dense random systems over $\mathbb{F}_2$ in which the number of equations is 1.5 times the number of variables, MQR(n, $1.5n$). These systems are solved at degrees such that the extended linearized system at a certain degree D has a number of rows is greater than the number of columns. This observation is applicable to all the MQR(n, $1.5n$) systems, from $n = 2$, except the system MQR(10, 15) which is solved at degree $D = 3$ with number of rows equal to 165 and number of columns equal to 176. By using MutantXL algorithm, see Algorithm 4.1, we observe that systems MQR(10, 15), MQR(20, 30) and MQR(30, 45) are solved at degree 3,

Algorithm 6.3 WMXL

Inputs
P: set of m quadratic polynomials.
$Limit$: number of trails.
Output
$Solution$: A solution of P=0.
Variables
$\mathcal{M}^{acaulay}$: a matrix whose entries are the coefficients of a system of multivariate polynomial equations in graded lex order.
$\mathcal{M}_{sq}^{acaulay}$: a square submatrix.
$\tilde{P}$: set of all polynomials that are included in the system.
D: the highest degree of $\tilde{P}$.
$solved$: a flag to indicate whether the system is solved or not.
$attempt$: a counter for the number of trails.
$Wsolution$: set of solutions generated by Wiedemann algorithm for the linearized system.
$Mutants$: set of lower degree polynomials.
Begin
Initialization()
$\{\mathcal{M}^{acaulay}, Solution, Wsolution, Mutants \leftarrow \varnothing$, $solved \leftarrow$ false, $\tilde{P} \leftarrow P$, $D \leftarrow 2$ $\}$
repeat
  $\mathcal{M}^{acaulay} \leftarrow$ Linearize($\tilde{P}$)
  if $nRows(\mathcal{M}^{acaulay}) \geq nCols(\mathcal{M}^{acaulay})$ **then**
    $attempt \leftarrow 1$
    repeat
      $\mathcal{M}_{sq}^{acaulay} \leftarrow$ Make_square($\mathcal{M}^{acaulay}$)
      $Wsolution \leftarrow$ Wiedemann($\mathcal{M}_{sq}^{acaulay}$)
      if $Wsolution \neq \varnothing$ **then**
        ($solved$,$Solution$) $\leftarrow$ Check_solution(P, $Wsolution$)
        if $solved$ **then**
          Return $(solution)$
        end if
      end if
      $attempt \leftarrow attempt + 1$
    until $(attempt \geq Limit)$
  end if
  $Mutants \leftarrow$ Generate_mutants($\mathcal{M}^{acaulay}$, D)
  if $Mutants \neq \varnothing$ **then**
    $\tilde{P} \leftarrow \tilde{P} \cup$ Extend($Mutants$, D)
  else
    $D \leftarrow D + 1$
    $\tilde{P} \leftarrow \tilde{P} \cup$ Extend(P, D)
  end if
until ($solved$)
End

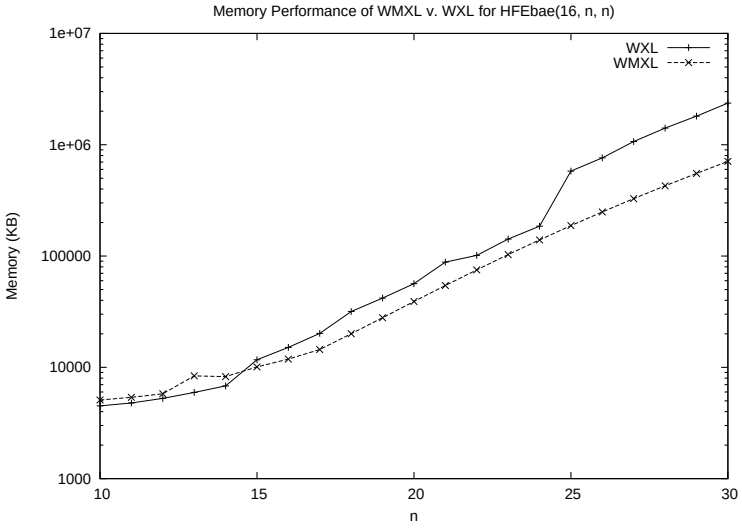

Figure 6.1: Memory Performance for WMXL versus WXL

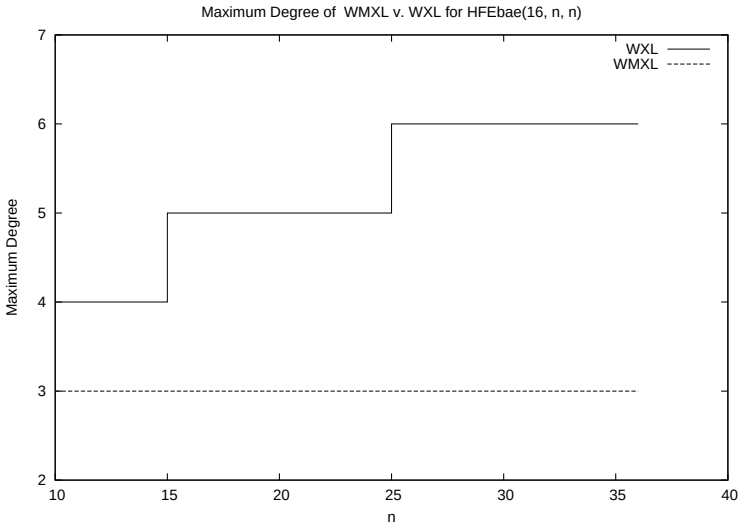

Figure 6.2: Maximum Degree for WMXL versus WXL

4 and 5, respectively using mutants while other systems are solved without using mutants. From these two observations, it is applicable to use WMXL to solve such systems. For the systems that are solved without the help of mutants, WMXL behaves like WXL. The usefulness of WMXL becomes very clear when it is impossible to store the dense matrix representation of an extended system.

Table 6.1 presents the memory and time performance for WMXL algorithm compared to MutantXL, F4 and MXL2 algorithms for MQR(n, $1.5n$). In this table, the first column "Sys" denotes the number of variables and the number of equations for each system. The maximum degree that is reached by each system is denoted by "D" which is the same for all solvers. The used memory in Megabytes (MiB) or Gigabytes (GB) and the execution time in seconds (S), minutes (M), hours (H) or days (D) for each solver are represented by "Mem" and "Time", respectively. The column "P" in the WMXL solver denotes the number of the used processors for the Wiedemann process. By using 128GB RAM, Table 6.1 shows that MutantXL can solve up to MQR(30, 45), F4 can solve up to MQR(34, 51), MXL2 can solve up to MQR(36, 54) and WMXL can solve up to MQR(40, 60). We are able to solve the five instances of MQR(n, $1.5n$) for $n = 2, 4, 6, 8$ at degree three. The next five instances for $n = 12, 14, 16, 18, 20$ are solved at degree four. The next group of instances for $n = 22, 24, 26, 28, 30$ is solved at degree five. The last group for $n = 32, 34, 36, 38, 40$ is solved at degree six. The fifth instance of each degree namely, MQR(10, 15), MQR(20, 30) or MQR(30, 45), is solved by using mutants.

The third scenario in this section is to use instances of HFE systems with univariate degree 288. These systems are solved at degree five. By using MXL2 as well as MutantXL to solve some instances of such systems, we obtain some observations. The first one is that the instance HFE(288, 25, 25) has 700 mutants at degree five. This number of mutants is increased by a constant value, namely 28, when increasing the number of variables by one. Figure 6.3 shows the relation between the number of variables n, from $n = 25$ to $n = 45$, and the number of extracted mutants for HFE systems with univariate degree 288. Therefore, the number of mutants for $n \geq 25$ variables could be computed by $28n$.

The second observation is that a number of linear mutants are generated after eliminating the new linearized system. This linearized system is produced from multiplying all the extracted degree four mutants then adding the new resulting multipliers to the system. These number of linear mutants help to solve the system at degree five without the need for more degree four mutants. Therefore, the HFE(288, n, n) systems are solved by one round degree four mutants.

The third observation is that the number of new generated linear mutants is always less than the number of variables except for the system HFE(288, 25, 25), the number of linear mutants are 25. Therefore, instead of solving using Wiedemann algorithm in the last step, we should generate linear mutants using Wiedemann algorithm then we add the generated linear mutants with the original quadratic polynomials and solve using MutantXL or MXL2 at degree two.

Table 6.2, shows the performance of KXL compared to F4 and MXL2 for HFE systems with univariate degree 288. In this table, the first column "n" denotes the number of variables. The used memory in Megabytes (MiB) or Gigabytes (GB) and the execution time in minutes (M), hours (H) or days (D) for each solver are represented by "Mem" and "Time", respectively. We use Magma's implementation of F4 version (V2.15-12). In these experiments, we use only 9 processors for KXL. This number of processors is used for generating mutants using Wiedemann algorithm while extending mutants, line 35 Algorithm 6.3, uses only one processor.

Under the fact that the running server has only 128GB RAM, F4 can solve up to the instance

Table 6.1: Performance of WMXL versus MutantXL, F4 and MXL2 for MQR(n, 1.5n)

Sys	D	MutantXL		$F4_{v2.16-8}$		MXL2		WMXL		P
		Mem	Time	Mem	Time	Mem	Time	Mem	Time	
MQR(10,15)*	3	3MiB	1S	9MiB	1S	3MiB	1S	2MiB	3S	1
MQR(12, 18)	4	3MiB	1S	9MiB	1S	3MiB	1S	3MiB	4S	1
MQR(14, 21)	4	4MiB	1S	10MiB	1S	3MiB	1S	4MiB	5S	1
MQR(16, 24)	4	5MiB	1S	12MiB	1S	4MiB	1S	6MiB	6S	1
MQR(18, 27)	4	8MiB	1S	15MiB	1S	5MiB	1S	8MiB	7S	1
MQR(20, 30)*	4	19MiB	2S	25MiB	2S	8MiB	2S	54MiB	1M	1
MQR(22, 33)	5	372MiB	3M	159MiB	4S	28MiB	5S	60MiB	1M	1
MQR(24, 36)	5	855MiB	4M	411MiB	13S	80MiB	14S	115MiB	2M	1
MQR(26, 39)	5	1.8GB	9M	884MiB	30S	227MiB	47S	195MiB	3M	1
MQR(28, 42)	5	3.7GB	21M	1.6GB	2M	817MiB	5M	300MiB	7M	1
MQR(30, 45)*	5	7.2GB	42M	3.7GB	27M	1.7GB	14M	3.3GB	2.1H	1
MQR(32, 48)	6			70.6GB	1.9H	13.9GB	6.4H	3.0GB	3.9H	9
MQR(34, 51)	6			128.5GB	7.7H	38.4GB	18.2H	5.3GB	8.5H	16
MQR(36, 54)	6					122.9GB	3.6D	8.8GB	10.1H	25
MQR(38, 57)	6							13.3GB	18.1H	36
MQR(40, 60)*	6							20.3GB	22.2H	81

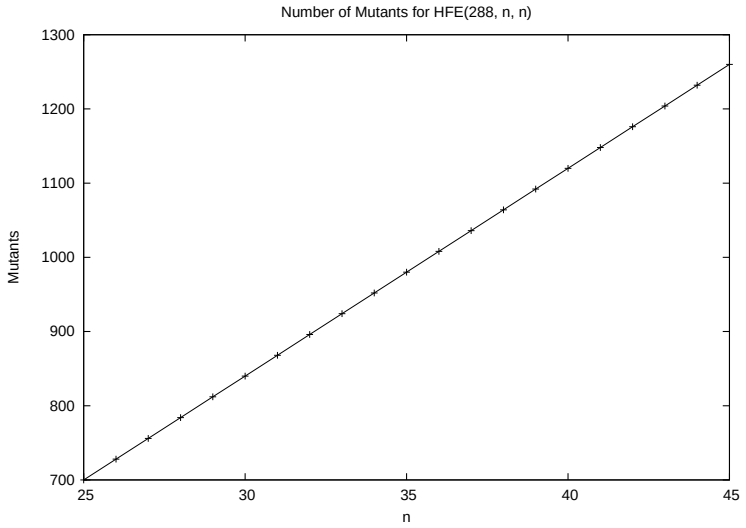

Figure 6.3: Number of Mutants for HFE(288, 25-45, 25-45)

HFE(288, 39, 39), MXL2 can solve up to HFE(288, 45, 45) and KXL can solve up to HFE(288, 50, 50). The memory performance for KXL using 9 processors is always better than F4. Starting from an instance of HFE(288, 36, 36), KXL outperforms MXL2 in terms of memory while for the instances of HFE(288, 45, 45) and HFE(288, 50, 50) KXL is better in terms of time and memory.

6.5 Conclusion

In this chapter, we show how to generate mutants using the block Wiedemann algorithm. We present the WMXL algorithm for solving systems of multivariate quadratic polynomial equations over $\mathbb{F}_2$. WMXL is an improvement of the XL algorithm that is based on the combination of the mutant strategy, the Wiedemann algorithm and the XL algorithm. For systems that are solved without the help of mutants, WMXL will behave like the WXL algorithm. A variant of WMXL which avoids Wiedemann algorithm as a solver is introduced. This variant is called KXL that is used for generating only mutants and the solving process is done using MutantXL or MXL2.

We introduce three different cases that provide a practical evidence for the effectiveness and efficiency of the WMXL and KXL. From a practical point of view, WMXL is better than Magma's F4 in terms of memory for HFE systems with univariate degree 288 and dense random systems with n variables and $1.5n$ equations. WMXL is better than MXL2 when it is impossible to store the dense matrix representation of a linearized extended system. Therefore, the gap between XL and F4 could be decreased by combining the concept of mutant and the parallelized Wiedemann algorithm in some specific cases.

Table 6.2: Performance of KXL versus F4 and MXL2 for HFE(288, n, n)

n	$F4_{v2.15}$		MXL2		KXL	
	Mem	Time	Mem	Time	Mem	Time
25	1.3GB	5M	428MiB	2M	1.4GB	16M
26	2.0GB	8M	618MiB	4M	1.8GB	22M
27	2.8GB	11M	1.2GB	9M	2.2GB	30M
28	4.1GB	16M	1.6GB	15M	2.9GB	52M
29	6.3GB	23M	2.2GB	21M	3.6GB	1.1H
30	9.0GB	32M	3.0GB	36M	4.5GB	1.7H
32	17.9GB	1.1H	5.3GB	1.4H	6.3GB	2.2H
34	30.8GB	2.4H	9.1GB	2.8H	9.2GB	4.5H
36	50.4GB	4.6H	15.2GB	6.2H	13.1GB	7.4H
38	81.5GB	8.1H	24.7GB	11.4H	18.4GB	12.6H
39	101.7GB	11.3H	31.2GB	14.8H	21.0GB	15.0H
40			39.1GB	19.4H	25.3GB	19.5H
45			111.8GB	3.6D	49.4GB	2.5D
50					94.3GB	8.8D

7 Conclusion and Future Work

In this thesis, we introduced different improvements of the XL algorithm. These improvements were used to solve several types of polynomial equation systems in the context of algebraic cryptanalysis. In this regard, the suggested improvements were able to compete with the F4 algorithm, the best known algorithm for solving systems of multivariate polynomial equations over finite fields.

In order to minimize the degree at which a system could be solved, we introduced the idea of extracting mutant polynomials. In the XL algorithm, these mutants are the low degree polynomials that appear after the linear algebra step. We stated the concept of a mutant. In this concept, when we extend a system to find a solution with a smaller degree, we give the priority to multiply mutants of lower degree. Based on this concept, we presented the MutantXL and MXL2 algorithms for solving systems of Boolean multivariate polynomial equations. MutantXL utilized mutants to improve XL while MXL2 applied two considerable improvements over $\mathbb{F}_2$ beside mutants. The MXL2 additional improvements were the use of necessary number of mutant and the partial enlargement technique. By using these two improvements, MXL2 was given the opportunity to consequently solve systems with smaller matrices size than XL and MutantXL. The practical evidences for the efficiency of MXL2 versus the F4 algorithm were presented by solving different types of systems. In particular, MXL2 outperformed F4 in terms of memory while for time performance, it is not always the case. In the context of algebraic cryptanalysis, MXL2 was used to break two multivariate public-key cryptosystems, namely Little Dragon Two and Poly-Dragon. MXL2 has shown to be more efficient than F4 for solving instances of both the Little Dragon Two and Poly-Dragon cryptosystems.

We considered the idea of using a sparse linear algebra solver instead of dense solvers in the XL algorithm. We represented the WXL algorithm which was an improvement of XL. WXL utilized the block Wiedemann algorithm over $\mathbb{F}_2$ as a solver instead of Gaussian elimination. We introduced the PWXL algorithm which was a parallelized version of WXL. The experimental results for both algorithms were presented based on dense random systems and instances of the HFE cryptosystem over $\mathbb{F}_2$. By using PWXL, we were able to solve instances of the HFE cryptosystem of univariate degree 4352 up to 37 equations in 37 variables. The instance with 37 equations in 37 variables was solved in almost 8 days using 81 processors, which was never done before by any known algebraic solver. The effectiveness of the PWXL algorithm was achieved by solving large systems where the ability to store the dense representation of these systems is impossible under the used resources.

We introduced the idea of combining the advantages of the improvements based on the concept of a mutant and the parallelized Wiedemann algorithm. The ability for MXL2 as well as MutantXL to solve structured systems at a degree smaller than XL is the benefit gained from mutants. The parallelization and sparse matrix representation for large systems are the advantages of using the Wiedemann algorithm. We discussed how to generate mutants using the block Wiedemann algorithm. We presented the WMXL algorithm which was a combination of the PWXL algorithm

and the mutant strategy. In WMXL, we used the block Wiedemann algorithm to generate mutants as well as to find a solution instead of Gaussian elimination. A variant of WMXL called KXL was suggested to solve some specific types of systems. Using block Wiedemann algorithm to generate mutants and using MutantXL to obtain a solution was the main idea of KXL. We presented three different cases that provided a practical evidences for the effectiveness and efficiency of WMXL. In particular, by using WMXL, we were able to solve a dense random system with 60 equations in 40 variables in almost 22 hours using 81 processors and less than 25GB RAM, which was impossible for F4 and MXL2.

Further work. In this thesis, we presented the design, implementation and practical evidences for the efficiency of the suggested improvements for the XL algorithm. The theoretical analysis of these improved algorithms is an important future work. The analysis of these algorithms provides theoretical estimates for the needed resources, such as time and storage, that are necessary to execute them. The combination of mutants and Involutive bases could be a future research direction which needs much time for the theoretical and experimental analysis.

While we presented promising efficiency for the suggested improvements for the XL algorithm, performance of the current versions is still below what would be desirable. In order to improve MXL2 time performance, we could compute the row echelon form (REF) using the PLS decomposition algorithm instead of the used M4RI algorithm. Another pressing area of improving MXL2 is to parallelize it. This parallelization could be done by performing parallel polynomials multiplication and performing parallel computation of the REF.

It would be interesting to investigate the possibility of implementing MXL2 over extension fields using the new M4RIE package which provides a solver over $\mathbb{F}_{2^e}$. In this regard, we could also use the Wiedemann algorithm to solve systems over extension fields. Before solving using the Wiedemann algorithm, we need to apply polynomials multiplication up to some degree over $\mathbb{F}_{2^e}$ and convert the coefficient matrix to a matrix over $\mathbb{F}_2$.

The bottleneck for the block Wiedemann algorithm is the matrix-vector multiplication process. A speedup in this operation could affect directly on the whole algorithm. Applying matrix-vector multiplications based on the structure of the matrices generated by XL could improve the efficiency of the PWXL algorithm and WMXL algorithm.

References

[1] Martin Albrecht. Algebraic Attacks on the Courtois Toy Cipher. Master's thesis, Universität Bremen, 2007.

[2] Martin Albrecht. *Algorithmic Algebraic Techniques and their Application to Block Cipher Cryptanalysis*. Dissertation, Royal Holloway, University of London, UK, 2010.

[3] Martin Albrecht and Gregory Bard. *The M4RI Library – Versions 20080511-20100817*. The M4RI Team, 2010.

[4] Martin Albrecht and Clément Pernet. Efficient Decomposition of Dense Matrices over GF(2). *Computing Research Repository (CoRR)*, abs/1006.1744, 2010.

[5] Martin Albrecht and John Perry. F4/5. Technical Report arXiv:1006.4933v2, October 2010. Available at `http://arxiv.org/abs/1006.4933v2`.

[6] Kazumaro Aoki, Jens Franke, Thorsten Kleinjung, Arjen K. Lenstra, and Dag Arne Osvik. A kilobit Special Number Field Sieve Factorization. In *Proceedings of the Advances in Crypotology 13th international conference on Theory and application of cryptology and information security*, ASIACRYPT'07, pages 1–12, Berlin, Heidelberg, 2007. Springer-Verlag.

[7] Frederik Armknecht and Matthias Krause. Algebraic Attacks on Combiners with Memory. In *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 162–175, Santa Barbara, California, USA, August 2003. Springer.

[8] Gwénolé Ars, Jean-Charles Faugère, Hideki Imai, Mitsuru Kawazoe, and Makoto Sugita. Comparison between XL and Gröbner Basis Algorithms. In *ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 338–353, Jeju Island, Korea, December 2004. Springer Berlin / Heidelberg.

[9] Gregory V. Bard. Accelerating Cryptanalysis with the Method of Four Russians. Cryptology ePrint Archive, Report 2006/251, 2006. Available at `http://eprint.iacr.org/`.

[10] Gregory V. Bard. *Algorithms for Solving Linear and Polynomial Systems of Equations over Finite Fields with Applications to Cryptanalysis*. Dissertation, University of Maryland, 2007.

[11] Gregory V. Bard. *Algebraic Cryptanalysis*. Springer, 2009.

[12] Gregory V. Bard. Matrix Inversion, (or LUP-Factorization) via the Method of Four Russians, in $\Theta(n^3/\log n)$ Time. The authors Web page, In submission. Available at `http://www-users.math.umd.edu/~bardg/m4ri.new.pdf`.

[13] Gregory V. Bard, Nicolas Tadeusz Courtois, and Chris Jefferson. Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over GF(2) via SAT-Solvers. Cryptology ePrint Archive, Report 2007/024, 2007.

[14] Magali Bardet, Jean-Charles Faugère, and Bruno Salvy. Complexity of Gröbner Basis Computation for Semi-regular Sequences over F_2 with Solutions in F_2. Technical report, Unité de Recherche INRIA Lorraine, 2003.

[15] O. Billet and J. Ding. Overview of Cryptanalysis Techniques in Multivariate Public Key Cryptography. In M.Sala et al. (Eds.), editor, *Grbner Bases, Coding, and Cryptography*, Lecture Notes in Computer Science, pages 263–284. Springer-Verlag, Berlin Heidelberg, 2009.

[16] Julia Borghoff, Lars R. Knudsen, and Mathias Stolpe. Bivium as a Mixed-Integer Linear Programming Problem. In *Cryptography and Coding '09: Proceedings of the 12th IMA International Conference on Cryptography and Coding*, pages 133–152, Berlin, Heidelberg, 2009. Springer-Verlag.

[17] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma Algebra System I: The User Language. *Journal of Symbolic Computation*, 24(3-4):235–265, 1997.

[18] Charles Bouillaguet, Pierre-Alain Fouque, Antoine Joux, and Joana Treger. A Family of Weak Keys in HFE (and the Corresponding Practical Key-Recovery). In Carlos Cid and Jean-Charles Faugère, editors, *Proceedings of the 2nd international conference on Symbolic Computation and Cryptography (SCC2010)*, pages 209–228, Royal Holloway, University of London, Egham - UK, June 2010. Available at `http://scc2010.rhul.ac.uk/scc2010-proceedings.pdf`.

[19] Michael Brickenstein and Alexander Dreyer. PolyBoRi: A Framework for Gröbner-basis Computations with Boolean Polynomials. *Journal of Symbolic Computation*, 44(9):1326–1345, 2009.

[20] Bruno Buchberger. *An Algorithm for Finding a Basis for the Residue Class Ring of a Zero-dimensional Polynomial Ring*. Dissertation, University of Innsbruck, 1965.

[21] Bruno Buchberger. Gröbner Bases: A Short Introduction for Systems Theorists. In *Computer Aided Systems Theory - EUROCAST 2001-Revised Papers*, pages 1–19, London, UK, 2001. Springer-Verlag.

[22] Johannes Buchmann, Stanislav Bulygin, Jintai Ding, Wael Said Abd Elmageed Mohamed, and Fabian Werner. Practical Algebraic Cryptanalysis for Dragon-Based Cryptosystems. In Swee-Huay Heng, Rebecca N. Wright, and Bok-Min Goi, editors, *Proceedings of the 9th International Conference on Cryptology and Network Security (CANS 2010)*, volume 6467 of *Lecture Notes in Computer Science*, pages 140–155. Springer, 2010.

[23] Johannes Buchmann, Daniel Cabarcas, Jintai Ding, and Mohamed Saied Emam Mohamed. Flexible Partial Enlargement to Accelerate Gröbner Basis Computation over $\mathbb{F}_2$. In *Proceedings of the 3rd Africacrypt Conference, (Africacrypt 2010), to be appear*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, May 2010.

[24] Johannes Buchmann, Jintai Ding, Mohamed Saied Emam Mohamed, and Wael Said Abd Elmageed Mohamed. MutantXL: Solving Multivariate Polynomial Equations for Cryptanalysis. In Helena Handschuh, Stefan Lucks, Bart Preneel, and Phillip Rogaway, editors, *Symmetric Cryptography*, number 09031 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2009. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany. Available at `http://drops.dagstuhl.de/opus/volltexte/2009/1945`.

[25] Jonathan F. Buss, Gudmund S. Frandsen, and Jeffery O. Shallit. The Computational Complexity of Some Problems of Linear Algebra. *J. Comput. Syst. Sci.*, 58:572–596, June 1999.

[26] John J. Cannon and Wieb Bosma. *Handbook of Magma Functions*. Edition 2.13 (2006).

[27] Fengjuan CHAI, Xiao-Shan GAO, and Chunming YUAN. A Characteristic Set Method for Solving Boolean Equations and Applications in Cryptanalysis of Stream Ciphers. *Journal of Systems Science and Complexity*, 21:191–208, 2008. 10.1007/s11424-008-9103-0.

[28] Baiqiang Chen. Strategies on Algebraic Attacks using SAT Solvers. In *ICYCS '08: Proceedings of the 2008 The 9th International Conference for Young Computer Scientists*, pages 2204–2209, Washington, DC, USA, 2008. IEEE Computer Society.

[29] Chia-Hsin Owen Chen, Ming-Shing Chen, Jintai Ding, Fabian Werner, and Bo-Yin Yang. Odd-Char Multivariate Hidden Field Equations, 2008.

[30] Jiali Choy, Huihui Yap, and Khoongming Khoo. An Analysis of the Compact XSL Attack on BES and Embedded SMS4. In *CANS '09: Proceedings of the 8th International Conference on Cryptology and Network Security*, pages 103–118, Berlin, Heidelberg, 2009. Springer-Verlag.

[31] Carlos Cid and Gaëtan Leurent. An Analysis of the XSL Algorithm. In Bimal Roy, editor, *ASIACRYPT*, volume 3788 of *Lecture Notes in Computer Science*, pages 333–352. Springer Berlin / Heidelberg, 2005.

[32] Carlos Cid, Sean Murphy, and Matthew Robshaw. *Algebraic Aspects of the Advanced Encryption Standard (Advances in Information Security)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[33] Stephen A. Cook. The complexity of Theorem-proving Procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM.

[34] Stephen A. Cook and D. Mitchel. Finding Hard Instances of The Satisfiability Problem: A Survey . In *Satisfiability Problem: Theory and Applications*, volume 25 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 1–17. American Mathematical Society, 1997.

[35] Don Coppersmith. Solving Homogeneous Linear Equations Over GF(2) via Block Wiedemann Algorithm. *Math. Comput.*, 62(205):333–350, 1994.

[36] Nicolas Courtois. The Security of Hidden Field Equations (HFE). In *Proceedings of the 2001 Conference on Topics in Cryptology: The Cryptographer's Track at RSA*, CT-RSA 2001, pages 266–281, London, UK, 2001. Springer-Verlag.

[37] Nicolas T. Courtois. Efficient Zero-Knowledge Authentication Based on a Linear Algebra Problem MinRank. In *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '01, pages 402–421, London, UK, 2001. Springer-Verlag.

[38] Nicolas T. Courtois and Gregory V. Bard. Algebraic Cryptanalysis of the Data Encryption Standard. In *Cryptography and Coding'07: Proceedings of the 11th IMA international conference on Cryptography and coding*, pages 152–169, Berlin, Heidelberg, 2007. Springer-Verlag.

[39] Nicolas T. Courtois, Gregory V. Bard, and David Wagner. Algebraic and Slide Attacks on KeeLoq. pages 97–115, 2008.

[40] Nicolas T. Courtois, Sean O'Neil, and Jean-Jacques Quisquater. Practical Algebraic Attacks on the Hitag2 Stream Cipher. In Pierangela Samarati, Moti Yung, Fabio Martinelli, and Claudio Ardagna, editors, *ISC '09: Proceedings of the 12th International Conference on Information Security*, Lecture Notes in Computer Science, pages 167–176, Berlin, Heidelberg, 2009. Springer-Verlag.

[41] Nicolas Tadeusz Courtois. Higher Order Correlation Attacks, XL Algorithm and Cryptanalysis of Toyocrypt. In *proceding of 5th International Conference on Information Security and Cryptology (ICISC)*, volume 2587 of *Lecture Notes in Computer Science*, pages 182–199, Seoul, Korea, November 2002. Springer-Verlag.

[42] Nicolas Tadeusz Courtois. Algebraic Attacks over GF(2^k), Application to HFE Challenge 2 and Sflash-v2. In *Public Key Cryptography (PKC 2004), 7th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2947 of *Lecture Notes in Computer Science*, pages 201–217. Springer, March 2004.

[43] Nicolas Tadeusz Courtois. Experimental Algebraic Cryptanalysis of Block Ciphers. http://www.cryptosystem.net/aes/toyciphers.html, 2007.

[44] Nicolas Tadeusz Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In *EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 392–407, Bruges, Belgium, May 2000. Springer.

[45] Nicolas Tadeusz Courtois and Jacques Patarin. About the XL Algorithm over GF(2). In *Proceedings of The Cryptographers' Track at the RSA Conference(CT-RSA)*, volume 2612 of *Lecture Notes in Computer Science*, pages 141–157, San Francisco, CA, USA, April 2003. Springer.

[46] Nicolas Tadeusz Courtois and Josef Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In Yuliang Zheng, editor, *Advances in Cryptology ASIACRYPT*

2002, volume 2501 of *Lecture Notes in Computer Science*, pages 267– 287. Springer Berlin / Heidelberg, 2002.

[47] Nicolas Tadeusz Courtois and Josef Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. Technical Report 0044, Cryptology ePrint Archive, 2002.

[48] David A. Cox, John Little, and Donal O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[49] Debapratim De, Abishek Kumarasubramanian, and Ramarathnam Venkatesan. Inversion Attacks on Secure Hash Functions using SAT Solvers. In *SAT'07: Proceedings of the 10th international conference on Theory and applications of satisfiability testing*, pages 377–382, Berlin, Heidelberg, 2007. Springer-Verlag.

[50] Peter Dembowski and T. G. Ostrom. Planes of Order n with Collineation Groups of Order n^2. *Mathematische Zeitschrift*, 103:239–258, 1968. 10.1007/BF01111042.

[51] Dhananjoy Dey, Prasanna Raghaw Mishra, and Indranath Sen Gupta. HF-hash : Hash Functions using Restricted HFE Challenge-1. *Computing Research Repository (CoRR)*, abs/0909.1392, 2009.

[52] Claus Diem. The XL Algorithm and a Conjecture from Commutative Algebra. In *ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 323–337, Jeju Island, Korea, December 2004. Springer Berlin / Heidelberg.

[53] Jintai Ding. Mutants and its Impact on Polynomial Solving Strategies and Algorithms. Privately distributed research note, University of Cincinnati and Technical University of Darmstadt, 2006.

[54] Jintai Ding. A New Variant of the Matsumoto-Imai Cryptosystem through Perturbation. In Feng Bao, Robert H. Deng, and Jianying Zhou, editors, *Public Key Cryptography - PKC 2004, 7th International Workshop on Theory and Practice in Public Key Cryptography*, volume 2947 of *Lecture Notes in Computer Science*, pages 305–318. Springer, 2004.

[55] Jintai Ding, Johannes Buchmann, Mohamed Saied Emam Mohamed, Wael Said Abdelmageed Mohamed, and Ralf-Philipp Weinmann. MutantXL. In *Proceedings of the 1st international conference on Symbolic Computation and Cryptography (SCC08)*, pages 16 – 22, http://www.cdc.informatik.tu–darmstadt.de/reports/reports/MutantXL_Algorithm.pdf, Beijing, China, April 2008. LMIB.

[56] Jintai Ding, Danial Cabarcas, Dieter Schmidt, Johannes Buchmann, and Stefan Tohaneanu. Mutant Gröbner Basis Algorithm. In *Proceedings of the 1st international conference on Symbolic Computation and Cryptography (SCC08)*, pages 23 – 32, Beijing, China, April 2008. LMIB.

[57] Jintai Ding, Jason E. Gower, and Dieter Schmidt. *Multivariate Public Key Cryptosystems (Advances in Information Security)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[58] Jintai Ding, Jason E. Gower, and Dieter S. Schmidt. Zhuang-Zi: A New Algorithm for Solving Multivariate Polynomial Equations over a Finite Field. In *PQCrypto 2006: Proceedings of the International Workshop on Post-Quantum Cryptography*, Katholieke University Leuven, Belgium, 2006.

[59] Jintai Ding, Lei Hu, Bo-Yin Yang, and Jiun-Ming Chen. Note on Design Criteria for Rainbow-Type Multivariates. Cryptology ePrint Archive, Report 2006/307, 2006. Available at `http://eprint.iacr.org/2006/307`.

[60] Jintai Ding and Dieter Schmidt. Cryptanalysis of HFEv and Internal Perturbation of HFE. In Serge Vaudenay, editor, *Public Key Cryptography-PKC 2005, 8th International Workshop on Theory and Practice in Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 288–301, Les Diablerets, Switzerland, January 2005. Springer.

[61] Jintai Ding and Dieter Schmidt. Rainbow, a New Multivariable Polynomial Signature Scheme. In John Ioannidis, Angelos D. Keromytis, and Moti Yung, editors, *Proceedings of Third International Conference on Applied Cryptography and Network Security (ACNS 2005)*, volume 3531 of *Lecture Notes in Computer Science*, pages 164–175, 2005.

[62] Jintai Ding and Dieter Schmidt. Mutant zhuang-zi algorithm. In *PQCrypto 2010: Proceedings of the Third International Workshop on Post-Quantum Cryptography*, volume 6061 of *Lecture Notes in Computer Science*, pages 28–40. Springer, 2010.

[63] Jintai Ding, Dieter Schmidt, and Fabian Werner. Algebraic Attack on HFE Revisited. In Tzong-Chen Wu, Chin-Laung Lei, Vincent Rijmen, and Der-Tsai Lee, editors, *Information Security, 11th International Conference, ISC 2008*, volume 5222 of *Lecture Notes in Computer Science*, pages 215–227. Springer, 2008.

[64] Jintai Ding, Christopher Wolf, and Bo-Yin Yang. ℓ-invertible Cycles for Multivariate Quadratic (MQ) Public Key Cryptography. In *Proceedings of the 10th international conference on Practice and theory in public-key cryptography*, PKC'07, pages 266–281, Berlin, Heidelberg, 2007. Springer-Verlag.

[65] Jintai Ding and Bo-Yin Yang. Multivariate Public Key Cryptography. In Daniel J et al. (Eds.) Bernstein, editor, *Post Quantum Cryptography*, pages 193–234. Springer Publishing Company, 2008.

[66] Vivien Dubois, Louis Granboulan, and Jacques Stern. An Efficient Provable Distinguisher for HFE. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming*, volume 4052 of *Lecture Notes in Computer Science*, pages 156–167. Springer Berlin / Heidelberg, 2006.

[67] Thomas Dullien. Algebraic Attacks Specialized for $\mathbb{F}_2$. Master's thesis, HGI, University of Bochum, Bochum, Germany, August 2008.

[68] Christian Eder and John Perry. F5C: a Variant of Faugère's F5 Algorithm with Reduced Gröbner Bases. Technical Report arXiv:0906.2967, Jun 2009. Comments: 31 pages, 4 tables.

[69] Niklas Eén and Niklas Sörensson. An Extensible SAT-solver. In *Theory and Applications of Satisfiability Testing, 6th International Conference*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[70] Tobias Eibach, Enrico Pilz, and Gunnar Völkel. Attacking Bivium using SAT solvers. In *SAT'08: Proceedings of the 11th international conference on Theory and applications of satisfiability testing*, pages 63–76, Berlin, Heidelberg, 2008. Springer-Verlag.

[71] Jeremy Erickson, Jintai Ding, and Chris Christensen. Algebraic Cryptanalysis of SMS4: Gröbner Basis Attack and SAT Attack Compared. In Donghoon Lee and Seokhie Hong, editors, *Information, Security and Cryptology - ICISC 2009, 12th International Conference*, volume 5984 of *Lecture Notes in Computer Science*, pages 73–86. Springer, 2009.

[72] Jean-Charles Faugére. A New Efficient Algorithm for Computing Gröbner Bases (F4). *Pure and Applied Algebra*, 139(1-3):61–88, June 1999.

[73] Jean-Charles Faugére. A New Efficient Algorithm for Computing Gröbner Bases without Reduction to Zero (F5). In *Proceedings of the 2002 international symposium on Symbolic and algebraic computation (ISSAC)*, pages 75 – 83, Lille, France, July 2002. ACM.

[74] Jean-Charles Faugère and Gwènolè Ars. Comparison of XL and Gröbner basis Algorithms over Finite Fields. Research Report RR-5251, INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE (INRIA), 2004.

[75] Jean-Charles Faugère and Antoine Joux. Algebraic Cryptanalysis of Hidden Field Equation (HFE) Cryptosystems using Gröbner Bases. In *Proceedings of the 23rd Annual International Cryptology Conference Advances in Cryptology - CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 44–60, Santa Barbara, California, USA, August 2003. Springer.

[76] Jean-Charles Faugère, Rune Ødegård, Ludovic Perret, and Danilo Gligoroski. Analysis of the MQQ Public Key Cryptosystem. In Swee-Huay Heng, Rebecca Wright, and Bok-Min Goi, editors, *Cryptology and Network Security*, volume 6467 of *Lecture Notes in Computer Science*, pages 169–183. Springer Berlin / Heidelberg, 2010.

[77] Adam Thomas Feldmann. *A Survey of Attacks on Multivariate Cryptosystems*. Dissertation, University of Waterloo, Ontario, Canada, 2005.

[78] Claudia Fiorini, Enrico Martinelli, and Fabio Massacci. How to Fake an RSA Signature by Encoding Modular Root Finding as a SAT Problem. *Discrete Appl. Math.*, 130(2):101–127, 2003.

[79] A. S. Fraenkel and Y. Yesha. Complexity of Problems in Games, Graphs and Algebraic Equations. *Discrete Applied Mathematics*, 1(1-2):15–30, 1979.

[80] Gerhard Koning Gans, Jaap-Henk Hoepman, and Flavio D. Garcia. A Practical Attack on the MIFARE Classic. In *CARDIS '08: Proceedings of the 8th IFIP WG 8.8/11.2 international conference on Smart Card Research and Advanced Applications*, pages 267–282, Berlin, Heidelberg, 2008. Springer-Verlag.

[81] Shuhong Gao, Yinhua Guan, and Frank Volny IV. A New Incremental Algorithm for Computing Gröbner Bases. In *ISSAC '10: Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation*, pages 13–19, New York, NY, USA, 2010. ACM.

[82] Shuhong Gao, Frank Volny IV, and Mingsheng Wang. A New Algorithm for Computing Groebner Bases. Cryptology ePrint Archive, Report 2010/641, 2010. Available at `http://eprint.iacr.org/`.

[83] Xiao-Shan GAO, Fengjuan CHAI, and Chunming YUAN. A Characteristic Set Method for Equations Solving in $\mathbb{F}_2$ and Applications in Cryptanalysis of Stream Ciphers. Technical Report 3, MM Research Preprints, 2006. Available at `http://www.mmrc.iss.ac.cn/pub/mm25.pdf/3.pdf`.

[84] Xiao-Shan Gao and Zhenyu Huang. A Characteristic Set Method for Equation Solving over Finite Fields. *ACM Commun. Comput. Algebra*, 42(3):149–150, 2008.

[85] Xiao-Shan Gao and Zhenyu Huang. A Characteristic Set Method for Equation Solving over Finite Fields. Technical Report 5, MM Research Preprints, 2008. Available at `http://www.mmrc.iss.ac.cn/pub/mm26.pdf/5-cs-Rq.pdf`.

[86] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[87] Danilo Gligoroski, Smile Markovski, and Svein Johan Knapskog. Multivariate Quadratic Trapdoor Functions based on Multivariate Quadratic Quasigroups. In *Proceedings of the American Conference on Applied Mathematics*, pages 44–49, Stevens Point, Wisconsin, USA, 2008. World Scientific and Engineering Academy and Society (WSEAS).

[88] Masahito Gotaishi and Shigeo Tsujii. HXL -a Variant of XL Algorithm Computing Gröbner Bases. In Jean-Charles Faugère, editor, *Proceedings of the 4th International SKLOIS Conference on Information Security and Cryptology (Special Track of Inscrypt 2008)*, pages 8–27. State Key Laboratory of Information Security (SKLOIS), 2008. Available at `http://www.inscrypt.cn/2008/index.html`.

[89] Louis Granboulan, Antoine Joux, and Jacques Stern. Inverting HFE Is Quasipolynomial. In *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference*, volume 4117 of *Lecture Notes in Computer Science*, pages 345–356. Springer, 2006.

[90] Lov K. Grover. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the 28th Annual ACM Symposium on Theory and Computing (STOC)*, pages 212–219, 1996.

[91] Omessad Hamdi, Ammar Bouallegue, and Sami Harari. Generalisation of Kipnis and Shamir Cryptanalysis of the HFE Public Key Cryptosystem. *World Academy of Science, Engineering and Technology*, 12:201–2004, 2005.

[92] Omessad Hamdi, Ammar Bouallegue, and Sami Harari. Hidden Field Equations Cryptosystem Performances. In *Proceedings of the IEEE International Conference on Computer Systems and Applications*, pages 308–311, Washington, DC, USA, 2006. IEEE Computer Society.

[93] Oscar H. Ibarra, Shlomo Moran, and Roger Hui. A Generalization of the Fast LUP Matrix Decomposition Algorithm and Applications. *Journal of Algorithms*, 3(1):45–56, 1982.

[94] Jean-Charles Faugère. Algebraic Cryptanalysis of HFE using Gröbner bases. Research Report RR-4738, INRIA, 2003.

[95] Claude-Pierre Jeannerod. LSP Matrix Decomposition Revisited. Technical Report 28, École Normale Supérieure de Lyon, 2006. Available at `http://www.ens-lyon.fr/LIP/Pub/Rapports/RR/RR2006/RR2006-28.pdf`.

[96] Wen Ji and Lei Hu. New Description of SMS4 by an Embedding over GF(2^8). In *INDOCRYPT'07: Proceedings of the cryptology 8th international conference on Progress in cryptology*, pages 238–251, Berlin, Heidelberg, 2007. Springer-Verlag.

[97] Xin Jiang, Jintai Ding, and Lei Hu. Kipnis-Shamir Attack on HFE Revisited. In *Information Security and Cryptology, Third SKLOIS Conference, Inscrypt 2007*, volume 4990 of *Lecture Notes in Computer Science*, pages 399–411. Springer, 2007.

[98] Antoine Joux, Sébastien Kunz-Jacques, Frédéric Muller, and Pierre-Michel Ricordel. Cryptanalysis of the Tractable Rational Map Cryptosystem. In *Public Key Cryptography - PKC 2005, 8th International Workshop on Theory and Practice in Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 258–274, Les Diablerets,Switzerland, 2005. Springer.

[99] Antoine Joux and Vanessa Vitse. A Variant of the F4 Algorithm. In Carlos Cid and Jean-Charles Faugère, editors, *Proceedings of the 2nd international conference on Symbolic Computation and Cryptography (SCC2010)*, pages 57 – 71, Royal Holloway, University of London, Egham - UK, June 2010. Available at `http://scc2010.rhul.ac.uk/scc2010-proceedings.pdf`.

[100] Dejan Jovanovic and Predrag Janicic. Logical Analysis of Hash Functions. In Bernhard Gramlich, editor, *Frontiers of Combining Systems, 5th International Workshop, FroCoS 2005*, volume 3717 of *Lecture Notes in Computer Science*, pages 200–215. Springer, 2005.

[101] Philipp Jovanovic and Martin Kreuzer. Algebraic Attacks using SAT-Solvers. In Carlos Cid and Jean-Charles Faugère, editors, *Proceedings of the 2nd international conference on Symbolic Computation and Cryptography (SCC2010)*, pages 7 – 17, Royal Holloway, University of London, Egham - UK, June 2010. Available at `http://scc2010.rhul.ac.uk/scc2010-proceedings.pdf`.

[102] Erich Kaltofen. Analysis of Coppersmith's Block Wiedemann Algorithm for the Parallel Solution of Sparse Linear Systems. In *Proceedings of the 10th International Symposium on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, AAECC-10, pages 195–212, London, UK, 1993. Springer-Verlag.

[103] Erich Kaltofen and A. Lobo. Distributed Matrix-Free Solution of Large Sparse Linear Systems over Finite Fields. *Algorithmica*, 24(3-4):331–348, 1999.

[104] Achim Kehrein and Martin Kreuzer. Computing Border Bases. *Journal of Pure and Applied Algebra*, 205(2):279 – 295, 2006.

[105] Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced Oil and Vinegar Signature Schemes. In *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'99, pages 206–222, Berlin, Heidelberg, 1999. Springer-Verlag.

[106] Aviad Kipnis and Adi Shamir. Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, pages 19–30, London, UK, 1999. Springer-Verlag.

[107] Aviad Kipnis and Adi Shamir. Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization. In *Proceedings of the 19th Annual International Cryptology Conference Advances in Cryptology - CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 19–30, Santa Barbara, California, USA, August 1999. Springer.

[108] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K. Lenstra, Emmanuel Thomé, Joppe W. Bos, Pierrick Gaudry, Alexander Kruppa, Peter L. Montgomery, Dag Arne Osvik, Herman J. J. te Riele, Andrey Timofeev, and Paul Zimmermann. Factorization of a 768-Bit RSA Modulus. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010, Proceedings of the 30th Annual Cryptology Conference*, volume 6223 of *Lecture Notes in Computer Science*, pages 333–350. Springer, 2010.

[109] Zbigniew Kokosiński. Algorithms for Unranking Combinations and their Applications. Technical Report 6, The University of Aizu, Department of Computer Software, 1995. Available at `http://riad.usk.pk.edu.pl/~zk/pubs/95-1-006.pdf`.

[110] Bonwook Koo, Hwanseok Jang, and Junghwan Song. The Condition of XSL for a Block Cipher. *Trends in Mathematics*, 8(1):69–75, 2005.

[111] Martin Kreuzer. Algebraic Attacks Galore! *Groups Complexity Cryptology*, 1(2):231–259, 2009.

[112] Rudolf Lidl and Harald Niederreiter. *Finite Fields*, volume 20 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge CB2 2RU, UK, 2 edition, 1997.

[113] Chu-Wee Lim and Khoongming Khoo. An Analysis of XSL Applied to BES. In Alex Biryukov, editor, *Fast Software Encryption, 14th International Workshop, FSE 2007*, volume 4593 of *Lecture Notes in Computer Science*, pages 242–253. Springer, 2007.

[114] Jeffrey T. Linderoth and Andrea Lodi. *"MILP Software", Encyclopedia for Operations Research*. Wiley, to appear 2010. Available at `http://homepages.cae.wisc.edu/~linderot/papers/Linderoth-Lodi-10.pdf`.

[115] Fabio Massacci and Laura Marraro. Towards the Formal Verification of Ciphers: Logical Cryptanalysis of DES. In *Proc. Third LICS Workshop on Formal Methods and Security Protocols, Federated Logic Conferences*, 1999.

[116] Fabio Massacci and Laura Marraro. Logical Cryptanalysis as a SAT Problem. *Journal of Automated Reasoning*, 24(1-2):165–203, 2000.

[117] Tsutomu Matsumoto and Hideki Imai. Public Quadratic Polynomial-tuples for Efficient Signature-Verification and Message-Encryption. In *Lecture Notes in Computer Science on Advances in Cryptology-EUROCRYPT'88*, pages 419–453, New York, NY, USA, 1988. Springer-Verlag New York, Inc.

[118] Jean-Francis Michon, Pierre Valarcher, and Jean-Baptiste Yunès. HFE and BDDs: A Practical Attempt at Cryptanalysis. In Keqin Feng, Harald Niederreiter, and Chaoping Xing, editors, *Proceedings of International Workshop on Coding, Cryptography and Combinatorics (CCC'03)*, page 11. Birkhauser, 2004.

[119] Ilya Mironov and Lintao Zhang. Applications of SAT Solvers to Cryptanalysis of Hash Functions. In Armin Biere and Carla P. Gomes, editors, *Theory and Applications of Satisfiability Testing - SAT*, volume 4121 of *Lecture Notes in Computer Science*, pages 102–115. Springer, 2006.

[120] T. Moh. A Public Key System with Signature and Master Key Functions. *Communications in Algebra*, 27(5):2207–2222, 1999.

[121] T. Moh. The Method of "Relinearization" of Kipnis and Shamir and It's Applications to TTM. http://citeseerx.ist.psu.edu, 1999.

[122] T. Moh. On the Method of "XL" and Its Inefficiency to TTM. Report 047, Cryptology ePrint Archive, January 2001.

[123] Mohamed Saied Emam Mohamed, Daniel Cabarcas, Jintai Ding, Johannes Buchmann, and Stanislav Bulygin. MXL3: An Efficient Algorithm for Computing Gröbner Bases of Zero-dimensional Ideals. In *Proceedings of The 12th international Conference on Information Security and Cryptology, (ICISC 2009)*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, December 2009.

[124] Mohamed Saied Emam Mohamed, Jintai Ding, and Johannes Buchmann. The Complexity Analysis of the MutantXL Family. Cryptology ePrint Archive, Report 2011/036, 2011. Available at `http://eprint.iacr.org/2011/036`.

[125] Mohamed Saied Emam Mohamed, Jintai Ding, Johannes Buchmann, and Fabian Werner. Algebraic Attack on the MQQ Public Key Cryptosystem. In *Proceedings of the 8th International Conference on Cryptology And Network Security, (CANS09)*, volume 5888 of *Lecture Notes in Computer Science*, pages 392–401, Kanazawa, Ishikawa, Japan, December 2009. Springer-Verlag, Berlin.

[126] Mohamed Saied Emam Mohamed, Wael Said Abd Elmageed Mohamed, Jintai Ding, and Johannes Buchmann. MXL2: Solving Polynomial Equations over GF(2) using an Improved Mutant Strategy. In *PQCrypto '08: Proceedings of the 2nd International Workshop on Post-Quantum Cryptography*, pages 203–215, Berlin, Heidelberg, 2008. Springer-Verlag.

[127] Pawel Morawiecki and Marian Srebrny. A SAT-based Preimage Analysis of Reduced KECCAK Hash Functions. Cryptology ePrint Archive, Report 2010/285, 2010. Available at http://eprint.iacr.org/2010/285.

[128] Sean Murphy and Maura B. Paterson. A Geometric View of Cryptographic Equation Solving. *Journal of Mathematical Cryptology*, 2(1):63 – 107, 2008.

[129] Sean Murphy and Maura B. Paterson. Geometric Ideas for Cryptographic Equation Solving in Even Characteristic. In *Cryptography and Coding '09: Proceedings of the 12th IMA International Conference on Cryptography and Coding*, pages 202–221, Berlin, Heidelberg, 2009. Springer-Verlag.

[130] Sean Murphy and M. J. B. Robshaw. Comments on the Security of the AES and the XSL Technique. http://www.isg.rhul.ac.uk/ sean/Xslbes8.ps, 2002.

[131] Nicolas T. Courtois and Gregory V. Bard. Algebraic Cryptanalysis of the Data Encryption Standard. In Steven D. Galbraith, editor, *Cryptography and Coding, 11th IMA International Conference*, volume 4887 of *Lecture Notes in Computer Science*, pages 152–169. Springer, 2007. Available at http://eprint.iacr.org/2007/024.

[132] Jacques Patarin. Cryptoanalysis of the Matsumoto and Imai Public Key Scheme of Eurocrypt'88. In *Advances in Cryptology - CRYPTO '95, 15th Annual International Cryptology Conference*, volume 963 of *Lecture Notes in Computer Science*, pages 248–261. Springer, 1995.

[133] Jacques Patarin. Asymmetric Cryptography with a Hidden Monomial. In *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '96, pages 45–60, London, UK, 1996. Springer-Verlag.

[134] Jacques Patarin. Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): two new families of Asymmetric Algorithms. In *Proceeding of International Conference on the Theory and Application of Cryptographic Techniques Advances in Cryptology- Eurocrypt*, volume 1070 of *Lecture Notes in Computer Science*, pages 33–48, Saragossa, Spain, May 1996. Springer.

[135] Jacques Patarin and Louis Goubin. Trapdoor One-way Permutations and Multivariate Polynominals. In *Proceedings of the First International Conference on Information and Communication Security*, pages 356–368, London, UK, 1997. Springer-Verlag.

[136] Jacques Patarin, Louis Goubin, and Nicolas Courtois. C^*_{-+} and HM: Variations Around Two Schemes of T. Matsumoto and H. Imai. In Kazuo Ohta and Dingyi Pei, editors, *Advances in Cryptology ASIACRYPT98*, volume 1514 of *Lecture Notes in Computer Science*, pages 35–50. Springer Berlin / Heidelberg, 1998.

[137] Buyun Qu and Lianhao Liu. An XSL Analysis on BES. In *ICYCS '08: Proceedings of the 2008 The 9th International Conference for Young Computer Scientists*, pages 1418–1423, Washington, DC, USA, 2008. IEEE Computer Society.

[138] Håvard Raddum. Cryptanalytic Results on TRIVIUM. Technical Report 039, eSTREAM, ECRYPT Stream Cipher Project, 2006.

[139] Håvard Raddum and Igor Semaev. New Technique for Solving Sparse Equation Systems. Cryptology ePrint Archive, Report 2006/475, 2006. Available at `http://eprint.iacr.org/2006/475`.

[140] Håvard Raddum and Igor Semaev. Solving MRHS Linear Equations. Cryptology ePrint Archive, Report 2007/285, 2007. Available at `http://eprint.iacr.org/2007/285`.

[141] Håvard Raddum and Igor Semaev. Solving Multiple Right Hand Sides Linear Equations. *Des. Codes Cryptography*, 49(1-3):147–160, 2008.

[142] A. Saikia Rajesh P. Singh and B.K. Sarma. Poly-Dragon: An efficient Multivariate Public Key Cryptosystem. Cryptology ePrint Archive, Report 2009/587, 2009. Available at `http://eprint.iacr.org/`.

[143] Sondre Rønjom and Håvard Raddum. On the Number of Linearly Independent Equations Generated by XL. In *SETA '08: Proceedings of the 5th international conference on Sequences and Their Applications*, pages 239–251, Berlin, Heidelberg, 2008. Springer-Verlag.

[144] Omonbek Salaev and Sudarshan Rao. Logical Cryptanalysis of CubeHash using a SAT Solver, 2009. Available at `http://www.cs.rit.edu/~ark/736/team/2/RaoSalaev.pdf`.

[145] Antoine Scemama. A Cryptanalysis of The Double-Round Quadratic Cryptosystem. In *ICISC'07: Proceedings of the 10th international conference on Information security and cryptology*, pages 27–36, Berlin, Heidelberg, 2007. Springer-Verlag.

[146] Patrick Schmidt. Solving Systems Of Multivariate Polynomial Equations using Mutants. http://www.cdc.informatik.tu-darmstadt.de/lehre/SS09/seminar/ausarbeitungen/Patrick

[147] A.J.M.Toon Segers. Algebraic Attacks from a Gröbner Basis Perspective. Master's thesis, Department of Mathematics and Computing Science, TECHNISCHE UNIVERSITEIT EINDHOVEN, Eindhoven, October 2004.

[148] Igor Semaev. On Solving Sparse Algebraic Equations over Finite Fields. *Des. Codes Cryptography*, 49(1-3):47–60, 2008.

[149] Igor Semaev. Sparse Algebraic Equations over Finite Fields. *SIAM Journal on Computing*, 39(2):388–409, 2009.

[150] Igor Semaev. Sparse Boolean Equations and Circuit Lattices. Cryptology ePrint Archive, Report 2009/252, 2009. Available at `http://eprint.iacr.org/2009/252`.

[151] Igor Semaev. Improved Agreeing-Gluing Algorithm. Cryptology ePrint Archive, Report 2010/140, 2010. Available at `http://eprint.iacr.org/2010/140`.

[152] Igor Semaev and Michal Mikuš. Methods to Solve Algebraic Equations in Cryptanalysis. *Tatra Mountains Mathematical Publications*, 45(1):107–136, 2010.

[153] Xiao shan Gao and Zhenyu Huang. Efficient Characteristic Set Algorithms for Equation Solving in Finite Fields and Application in Analysis of Stream Ciphers. Cryptology ePrint Archive, Report 2009/637, 2009. Available at `http://eprint.iacr.org/2009/637`.

[154] Mitsumari Shigeo. Hotaru. http://cvs.sourceforge.jp/cgi-bin/viewcvs.cgi/hotaru/hotaru/hfe25-96?view=markup.

[155] Peter Williston Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *SFCS '94: Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, Washington, DC, USA, 1994. IEEE Computer Society.

[156] Peter Williston Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Journal on Computing*, 26(5):1484 – 1509, 1997.

[157] Rajesh P Singh, Anupam Saikia, and B. K. Sarma. Little Dragon Two: An Efficient Multivariate Public Key Cryptosystem. *International Journal of Network Security and Its Applications (IJNSA)*, 2(2):1–10, 2010.

[158] Rajesh P. Singh, Anupam Saikia, and B. K. Sarma. Poly-Dragon: An efficient Multivariate Public Key Cryptosystem. *Journal of Mathematical Cryptology*, 4(4):349 – 364, 2011.

[159] Simon Singh. *The Code Book: How to Make It, Break It, Hack It, Crack It*. Delacorte Press, New York, NY, USA, 2001.

[160] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT Solvers to Cryptographic Problems. In *SAT '09: Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, pages 244–257, Berlin, Heidelberg, 2009. Springer-Verlag.

[161] W. A. Stein et al. *Sage Mathematics Software (Version 4.2.1)*. The Sage Development Team, 2009. Available at `http://www.sagemath.org`.

[162] Makoto Sugita, Mitsuru Kawazoe, and Hideki Imai. Relation between the XL Algorithm and Gröbner Basis Algorithms. *Transactions on Fundamentals of Electronics, Communications and Computer Sciences (IEICE)*, E89-A(1):11–18, January 2006.

[163] Xijin Tang and Yong Feng. A New Efficient Algorithm for Solving Systems of Multivariate Polynomial Equations. Technical Report 312, Cryptology ePrint Archive, 2005.

[164] Enrico Thomae and Christopher Wolf. Unreval XL and its variants. Cryptology ePrint Archive, Report 2010/596, 2010. Available at `http://eprint.iacr.org/`.

[165] Emmanuel Thomé. Subquadratic Computation of Vector Generating Polynomials and Improvement of the Block Wiedemann Algorithm. *J. Symb. Comput.*, 33(5):757–775, 2002.

[166] Ilia Toli. Cryptanalysis of HFE. *Computing Research Repository (CoRR)*, cs.CR/0305034, 2003.

[167] Carlo Traverso. Gröbner Trace Algorithms. In *ISAAC '88: Proceedings of the International Symposium ISSAC'88 on Symbolic and Algebraic Computation*, pages 125–138, London, UK, 1989. Springer-Verlag.

[168] William J. Turner. A Block Wiedemann Rank Algorithm. In *ISSAC '06: Proceedings of the 2006 international symposium on Symbolic and algebraic computation*, pages 332–339, New York, NY, USA, 2006. ACM.

[169] G. Villard. Further Analysis of Coppersmith's Block Wiedemann Algorithm for the Solution of Sparse Linear Systems (extended abstract). In *Proceedings of the 1997 international symposium on Symbolic and algebraic computation*, ISSAC '97, pages 32–39, New York, NY, USA, 1997. ACM.

[170] Lih-Chung Wang and Fei-Hwang Chang. Revision of tractable rational map cryptosystem. Cryptology ePrint Archive, Report 2004/046, 2004. Available at `http://eprint.iacr.org/2004/046.pdf`.

[171] Lih-Chung Wang, Yuh-Hua Hu, Feipei Lai, Chun yen Chou, and Bo-Yin Yang. Tractable Rational Map Signature. In *Public Key Cryptography (PKC 2005), The 8th International Workshop on Theory and Practice in Public Key Cryptography*, volume 3386 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2005.

[172] Lih-Chung Wang, Bo-Yin Yang, Yuh-Hua Hu, and Feipei Lai. A "Medium-Field" Multivariate Public-Key Encryption Scheme. In *CT-RSA 2006: Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006*, volume 3860 of *Lecture Notes in Computer Science*, pages 132–149. Springer, 2006.

[173] Fabian Werner. HFE Solved or Saved? Master's thesis, Technischen Universität Darmstadt, Darmstadt, Germany, 2007.

[174] Douglas H. Wiedemann. Solving Sparse Linear Equations over Finite Fields. *IEEE Trans. Inf. Theor.*, 32(1):54–62, 1986.

[175] Christopher Wolf. Hidden Field Equations (HFE) Variations and Attacks. Master's thesis, Department of Mathematics, University of College Cork (UCC), Cork, Ireland, 2002.

[176] Christopher Wolf. Efficient Public Key Generation for HFE and Variations. In Ed Dawson and Wolfgang Klemm, editors, *Cryptographic Algorithms and their Uses*, pages 78–93. Queensland University of Technology, 2004.

[177] Christopher WOLF. *Multivariate Quadratic Polynomials in Public Key Cryptography*. Dissertation, Katholieke Universiteit Leuven, 2005.

[178] Christopher Wolf, Patrick Fitzpatrick, Simon N. Foley, and Emanuel Popovici. HFE in Java: Implementing Hidden Field Equations for Public Key Cryptography. In *Proceedings of the 2002 Irish Signals and Systems Conference Cork (ISSC 2002)*, pages –, June 2002.

[179] Christopher Wolf and Bart Preneel. Asymmetric Cryptography: Hidden Field Equations. In *Proceedings of the 4th European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS 2004)*, 2004.

[180] Christopher Wolf and Bart Preneel. Taxonomy of Public Key Schemes based on the Problem of Multivariate Quadratic Equations. Cryptology ePrint Archive, Report 2005/077, 2005. Available at `http://eprint.iacr.org/`.

[181] L. Xiao. Applicability of XSL attacks to block ciphers. *Electronics Letters*, 39(25):1810–1811, 2003.

[182] Bo-Yin Yang and Jiun-Ming Chen. All in the XL Family: Theory and Practice. In *Proceedings of 7th International Conference on Information Security and Cryptology (ICISC)*, volume 3506 of *Lecture Notes in Computer Science*, pages 67–86, Seoul, Korea, December 2004. Springer.

[183] Bo-Yin Yang and Jiun-Ming Chen. Theoretical Analysis of XL over Small Fields. In *ACISP 2004*, volume 3108 of *Lecture Notes in Computer Science*, pages 277–288, Sydney, Australia, July 2004. Springer.

[184] Bo-Yin Yang and Jiun-Ming Chen. XL: A Brief on the State of the Art. In *Proceedings of Chinese (Taipei) Cryptology and Info. Sec. Assoc. conference (CCISA)*, June 2004.

[185] Bo-Yin Yang, Owen Chia-Hsin Chen, Daniel J. Bernstein, and Jiun-Ming Chen. Analysis of QUAD. In *Fast Software Encryption: 14th International Workshop, (FSE 2007)*, Lecture Notes in Computer Science, pages 290–308, Luxembourg, Luxembourg, March 2007. Springer-Verlag.

[186] Feng Yuan, Yu pu Hu, Yan Wang, and Hai wen Ou. Cryptanalysis of Dragon Scheme. *The Journal of China Universities of Posts and Telecommunications*, 17(4):80–87, 2010.

[187] Xiaoyi Zhou, Jixin Ma, Wencai Du, Bo Zhao, Mingrui Chen, and Yongzhe Zhao. Cryptanalysis of the Bisectional MQ Equations System. *Computer and Information Technology, International Conference on*, 0:1038–1043, 2010.

[188] A. I. Zobnin. Generalization of the F5 Algorithm for Calculating Gröbner Bases for Polynomial Ideals. *Program. Comput. Softw.*, 36(2):75–82, 2010.

Appendix: Definitions

In this Appendix, we present the basic definitions that are used throughout this thesis. For more details we refer the reader to [48, 112].

Definition .1. *An* **Abelian group** *is a set A with a binary operation $\bullet$ satisfying the following conditions:*

- *For all $a, b, c \in A$, the associative law holds i.e.*
  $a \bullet (b \bullet c) = (a \bullet b) \bullet c$.
- *For all $a \in A$, there exist identity element $e \in A$ such that*
  $a \bullet e = a$, $a \bullet e = a$
- *For any $a \in A$, there exists inverse element $b \in A$ such that*
  $a \bullet b = e$, $a \bullet b = e$.
- *For all $a, b, c \in A$, the commutative law holds i.e.*
  $a \bullet b = b \bullet a$.

Definition .2. *A* **Field** $\mathbb{F}$ *is a set with two binary operations $+$ and $*$ such that*

- $\mathbb{F}$ *is an Abelian group with respect to addition $+$.*
- $\mathbb{F}\backslash\{0\}$ *is an Abelian group with respect to addition $*$.*
- *Distributive law holds for all a,b and c in $\mathbb{F}$*
  a(b + c) = a*b + a*c*
  *(a+b)*c = a*c + b*c*

If the number of elements in $\mathbb{F}$ is finite, we call $\mathbb{F}$ a finite field and denote it $\mathbb{F}_n$, where n is the number of elements in $\mathbb{F}$

Definition .3. *A* **monomial** *in the n variables $x_1, \ldots, x_n$ is a product of the form $x_1^{\alpha_1}.x_2^{\alpha_2} \ldots x_n^{\alpha_n}$, where all the exponents $\alpha_1, \ldots, \alpha_n$ are nonnegative integers.*

Definition .4. *A* **polynomial** *f in the n variables $x_1, \ldots, x_n$ with coefficients in a field K is a finite linear combination (with coefficients in K) of monomials. i.e.*

$$f = \sum_{\alpha} a_\alpha x^\alpha, a_\alpha \in K$$

where the sum is over a finite number of n-tuples $\alpha = (\alpha_1, \ldots, \alpha_n)$.

Definition .5. *A* **polynomial ring** *is the set of all polynomials in the n variables $x_1, \ldots, x_n$ with coefficients in a field K denoted by $K[x_1, \ldots, x_n]$.*

Definition .6. *An* **ideal** *is a subset I such that $I \subseteq K[x_1, \ldots, x_n]$, $I \neq \emptyset$ if*

- *$f, g \in I$ implies that $f + g \in I$.*
- *$f \in I$ and $h \in K[x_1, \ldots, x_n]$ implies that $hf \in I$.*

Definition .7. *A* **monomial ordering** *denoted by $<$ on $K[x_1, \ldots, x_n]$ is a total ordering on the set of monomials of $K[x_1, \ldots, x_n]$ having two additional properties:*

- *The ordering respects multiplication. i.e.*
  whenever $Mo_1 > Mo_2$ then $Mo_3 Mo_1 > Mo_3 Mo_2$ where Mo_1, Mo_2, Mo_3 are monomials.
- *Every non-empty set of monomials has a minimal element. i.e.*
  $Mo >= 1$ for every monomial Mo.

Definition .8. *Let $f = \sum_\alpha a_\alpha x^\alpha, a_\alpha \in K$ be a nonzero polynomial in $K[x_1, \ldots, x_n]$ and let $>$ be a monomial ordering then:*

- *The* **degree** *of f denoted by deg is $deg(f) = \max(\alpha : a_\alpha \neq 0)$, with respect to $>$.*
- *The* **leading coefficient** *of f denoted by LC of f is $LC(f) = a_{deg(f)} \in K$.*
- *The* **leading monomial** *of f denoted by LM is $LM(f) = x^{deg(f)}$.*
- *The* **leading term** *of f denoted by LT is $LT(f) = LC(f).LM(f)$.*

Definition .9. *A* **Lexicographical ordering** *denoted by $>_{lex}$ on $K[x_1, \ldots, x_n]$ with*

$$x_1 >_{lex} x_2 >_{lex} \cdots >_{lex} x_n$$

is defined as follows: for two non-equal monomials

$x_1^{\alpha_1} \ldots x_n^{\alpha_n} >_{lex} x_1^{\beta_1} \ldots x_n^{\beta_n}$ *if and only if $\alpha_i > \beta_i$*
where i is the smallest number in $\{1, \ldots, n\}$ for which $\alpha_i \neq \beta_i$.

Definition .10. *A* **Graded lexicographic ordering** *denoted by $>_{grlex}$ on $K[x_1, \ldots, x_n]$ with*

$$x_1 >_{grlex} x_2 >_{grlex} \cdots >_{grlex} x_n$$

is defined as follows:
$x_1^{\alpha_1} \ldots x_n^{\alpha_n} >_{grlex} x_1^{\beta_1} \ldots x_n^{\beta_n}$ *if and only if* $\mid \alpha \mid = \sum_{i=1}^n \alpha_i > \mid \beta \mid = \sum_{i=1}^n \beta_i$, *or*
$\mid \alpha \mid = \mid \beta \mid$ *and* $\alpha >_{lex} \beta$.

Definition .11. *A* **Göbner basis** *for an ideal I is A set of non-zero polynomials $G = g_1, \ldots, g_t$ contained in I if and only if for all $f \in I$ such that $f \neq 0$, there exist $i \in \{1, \ldots, t\}$ such that $LM(g_i)$ divides $LM(f)$.*

Definition .12. *A* **Row Echelon Form** *for a matrix* **M** *is a matrix with the following properties:*

- *All nonzero rows are above any rows of all zeros.*
- *Each leading entry of a row is in a column to the right of the leading entry of the row above it.*
- *All entries in a column below a leading entry are zeros.*
- *The first nonzero number in any row is a one.*

Definition .13. *A* **Reduced Row Echelon Form** *for a matrix* **M** *is a matrix in* **Row Echelon Form** *and each leading coefficient of a non zero row is the only non zero element in its column.*

Definition .14. (PLUQ decomposition). *Any matrix* **M** *with dimensions* $d1 \times d2$ *and rank* r *is in a PLUQ decomposition if* **M** *can be written as* **M** = $PLUQ$ *where* P *and* Q *are two permutation matrices of dimensions* $d1 \times d1$ *and* $d2 \times d2$*, restrictively.* L *is an* $d1 \times r$ *unit lower triangular and* U *is an* $r \times d2$ *upper triangular matrix.*

Definition .15. (PLS decomposition). *Any matrix* **M** *with dimensions* $d1 \times d2$ *and rank* r *is in a PLS decomposition if* **M** *can be written as* **M** = PLS *where* P *is a permutation matrix of dimensions* $d1 \times d1$*,* L *is an* $d1 \times r$ *unit lower triangular and* S *is an* $r \times d2$ *upper triangular matrix except that its columns are permuted.*

Definition .16. *The* **minimal polynomial** *of an* $N \times N$ *matrix* **A** *over a finite field* $\mathbb{F}$ *is the monic polynomial* $P(x)$ *over* $\mathbb{F}$ *of least degree such that* $P(\mathbf{A}) = 0$*.*

Definition .17. *Given an* $N \times N$ *matrix* **A** *over a finite field* $\mathbb{F}$*, and a column vector* $v \in \mathbb{F}^n$*, the subspace of* $\mathbb{F}^n$ *spanned by* $\{v, Av, \ldots, A^i v, A^{i+1}v, \ldots\}$ *is called the* **krylov subspace** *of* **A** *and* v*.*

Definition .18. *The sequence* $\{a_i\}_{i=0}^{\infty}$ *is linearly generated (recurrent) over a finite field* $\mathbb{F}$ *if there exists* $n \in \mathbb{N}$ *and* $f_0, \ldots, f_n$ *with* $f_n \neq 0$ *such that,*
$\sum_{0 \leq j \leq n} f_j a_{i+j} = f_n a_{i+n} + \cdots + f_0 a_i = 0$
for $i \in \mathbb{N}$*. The polynomial* $f = \sum_{0 \leq j \leq n} f_j x^j \in \mathbb{F}[x]$ *of degree* n *is called a* **generating polynomial** *of* $\{a_i\}_{i=0}^{\infty}$*.*